Ana Catarina Gralha de Almeida

Licenciada em Engenharia Informática

Avaliação da Qualidade em Modelos de Requisitos *i**

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática

Orientador : Prof. Dr. Miguel Carlos Pacheco Afonso Goulão,
Prof. Auxiliar, Universidade Nova de Lisboa

Co-orientador : Prof. Dr. João Baptista da Silva Araújo Júnior,
Prof. Auxiliar, Universidade Nova de Lisboa

Júri:

Presidente: Prof. Dra. Carla Maria Gonçalves Ferreira

Arguente: Prof. Dr. José Alberto Rodrigues Pereira Sardinha

Vogal: Prof. Dr. Miguel Carlos Pacheco Afondo Goulão

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Setembro, 2014

Avaliação da Qualidade em Modelos de Requisitos *i

Em memória de Maria Madalena Ribeiro Lopes Gralha e
Joaquim de Oliveira Gralha.

Agradecimentos

Quero começar por agradecer aos meus orientadores, Professores Miguel Goulão e João Araújo, por todas as sugestões, críticas, e disponibilidade que demonstraram ao longo da realização desta dissertação. Com o seu apoio e orientação, o trabalho tornou-se muito mais fácil e interessante.

À Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, pelas condições oferecidas aos seus alunos, e pela formação sólida que lhes oferece. Ao Departamento de Informática, que se tornou a minha segunda casa nestes últimos anos, pela bolsa de investigação com componente de apoio letivo.

À Jenniffer Horkoff, pelas sugestões e ajudas sobre o *framework i**. À Professora Carla Silva, ao Diego Dermeval, e ao Carlos Lima, da Universidade Federal de Pernambuco, e ao Jorgen Engmann, da University College London, pela disponibilização de alguns dos casos de estudo necessários para a parte de avaliação presente nesta dissertação.

Aos meus companheiros de aventura, que partilharam comigo a sala, as brincadeiras, os momentos de diversão e de trabalho intenso, os desesperos e o alívio final. Ao Diogo Cordeiro, por todos estes anos. Ao Gabriel Marcondes, João Claro, Hélder Marques e Hélder Gregório. Aos que apareciam de vez em quando, normalmente depois de almoço ou quando o dia já estava quase a terminar: João Espada, André Grossinho, Gonçalo Tavares e João Ferreira. Ao Andy Gonçalves, que esteve sempre presente, embora de uma mais forma virtual.

Ao Ricardo Santos que, apesar de se encontrar a 2000 quilómetros de distância, sempre teve uma palavra amiga a dizer. À Joana Gato, por 20 anos de amizade. Ao Hugo Cabrita e ao Tiago Valente, pela preocupação demonstrada.

Um agradecimento muito especial ao João Silva, por todo o incentivo e paciência. Por me acalmar nos momentos de maior tensão, por me ajudar no que estava ao seu alcance, e por sempre acreditar no meu sucesso. Pela compreensão nos meus dias de mau humor, pelo apoio quando mais precisei dele, e por todos os momentos de pausa e diversão.

Por último mas não menos importante, quero agradecer à minha mãe, pela educação e por todo o seu apoio e incentivos fornecidos ao longo dos anos.

Resumo

As abordagens de engenharia de requisitos orientada a objetivos oferecem mecanismos que ajudam na elicitação e modelação de requisitos. A presente dissertação centra-se no *framework i**, uma das mais importantes linguagens de especificação de modelos de requisitos orientados a objetivos, que conta com grupos de trabalho em mais de vinte países e com sete edições de *workshops* internacionais.

Existem alguns problemas que podem afetar a qualidade de modelos *i**. A sua criação pode resultar em modelos complexos e/ou incompletos, assim como incorretos devido ao mau uso da notação do *i**. Estes problemas contribuem para a existência de dificuldades na gestão e compreensão dos sistemas de *software* o que, por sua vez, leva a um aumento dos custos de desenvolvimento. A qualidade dos modelos deve, então, ser uma preocupação constante, por forma a que sejam construídos modelos corretos, completos e cuja complexidade acidental seja minimizada. Não obstante a sua importância, a gestão eficaz destes três aspetos é, ainda, um problema em aberto.

Nesta dissertação é proposto um conjunto de métricas para dar suporte à avaliação quantitativa dos atributos de qualidade complexidade, completude e correção nos modelos de requisitos *i**. As métricas são definidas seguindo a abordagem *Goal-Question-Metric*, sendo definidas tanto de modo informal, em língua natural, como formalmente, em OCL (*Object Contraint Language*), e incorporadas numa ferramenta de modelação *i** para que possam ser recolhidas de modo automático. As métricas são avaliadas experimentalmente, através de um conjunto de casos de estudo reais e académicos, e teoricamente, através de propriedades de Weyuker.

Palavras-chave: Engenharia de Requisitos Orientada a Objetivos, Métricas de Complexidade, Métricas de Completude, Métricas de Correção, Qualidade do *Software*, *Framework i**

Abstract

Goal-oriented requirements engineering approaches provide mechanisms to help eliciting and modelling requirements. This thesis focuses on the *i** framework, one of the most important languages for specifying goal-oriented requirements models, which has working groups in over twenty countries and has seven editions of international workshops.

There are some problems that can affect the quality of *i** models. Their creation can result in complex and/or incomplete models, as well as incorrect ones due to the misuse of the *i** notation. These problems contribute to difficulties in the management and understanding of software systems, which in turn leads to increased development costs. The quality of the models should then be a permanent concern, in order to construct models that are correct, complete and whose accidental complexity is minimized. Despite its importance, the effective management of these three aspects is still an open problem.

In this thesis, we propose a set of metrics to support the quantitative assessment of the quality attributes complexity, completeness and correctness of *i** models. Those metrics are defined through the Goal-Question-Metric approach, being specified both informally, in natural language, as well as formally, with OCL (Object Constraint Language), and incorporated in an *i** modelling tool so that they can be automatically collected. The metrics are evaluated both experimentally, through a set of real-world and academic case studies, as well as theoretically, through Weyuker's properties.

Keywords: Goal-oriented Requirements Engineering, Requirements Complexity Metrics, Requirements Completeness Metrics, Requirements Correctness Metrics, Software Quality, *i** Framework

Conteúdo

Lista de Figuras

Lista de Tabelas

Listagens

1

Introdução

Este capítulo contém uma introdução ao trabalho realizado nesta dissertação, e começa por apresentar uma descrição do seu foco e contexto, e quais as motivações do mesmo. Em seguida, apresentam-se os objetivos e as principais contribuições previstas. Termina-se com a apresentação da estrutura do presente documento.

1.1 Descrição e Contexto

A engenharia de *software* pode ser descrita como a aplicação e o estudo de abordagens sistemáticas, disciplinadas e quantificáveis para o desenvolvimento, operação e manutenção do *software*, ou seja, a aplicação de engenharia ao *software* [Abr+04]. Por outras palavras, é uma área que tem como preocupação todos os aspetos da produção de sistemas de *software*, desde a fase inicial de especificação de requisitos até à manutenção do sistema depois de o mesmo já se encontrar em utilização [Som10]. Esta área pode ser dividida em várias sub-disciplinas, tais como: requisitos, desenho, construção, teste, manutenção, gestão de configurações, gestão de engenharia, processos de engenharia, ferramentas e métodos de engenharia, e qualidade [Abr+04].

Dentro das diversas sub-disciplinas apresentadas, este trabalho enquadra-se nas áreas de requisitos de *software* e qualidade. A engenharia de requisitos tem como principal objetivo a identificação e análise dos requisitos de um sistema, assim como as restrições que este deve respeitar, e inclui: elicitação, especificação, validação e gestão de requisitos [KS04]. Os requisitos são definidos como propriedades que devem ser especificadas por forma a resolver algum problema do mundo real. Existem vários paradigmas que são utilizados para a sua estruturação e representação, tais como: objetivos, objetos, agentes, pontos de vista, cenários e aspetos [Som10].

Este trabalho tem como foco a engenharia de requisitos orientada a objetivos (em inglês *Goal-Oriented Requirements Engineering* - GORE [Lam01]). A especificação desses requisitos é suportada por diversas metodologias, sendo as principais o *framework i** (istar) [Yu95; Yu97], o *framework* NFR (do inglês *Non-Functional Requirements*) [TC99] e a metodologia KAOS (do inglês *Knowledge Acquisition in autOmated Specification*) [DLF93; Lam14]. Estas metodologias ajudam na decomposição dos requisitos com base em objetivos, permitindo compreender melhor o sistema que se deseja modelar. De entre as metodologias apresentadas, o presente trabalho centra-se no *framework i**, uma das mais importantes linguagens de especificação de modelos de requisitos orientados a objetivos, que conta com grupos de trabalho em mais de 20 (vinte) países e com 7 (sete) edições de *workshops* internacionais [wik14f].

Este trabalho também se foca na qualidade do *software* e na sua avaliação, em particular na qualidade dos artefactos que correspondem aos modelos de requisitos. A qualidade do *software* é o grau em que um sistema, componente ou processo cumpre requisitos inicialmente especificados, estando diretamente relacionada com a qualidade do processo de desenvolvimento [Gal04]. A qualidade do *software* pode ser avaliada através de atributos de qualidade, tais como a sua complexidade, completude, correção, usabilidade, segurança, entre outros [Som10]. A presente dissertação centra-se na complexidade, na completude e na correção dos modelos de requisitos *i**.

1.2 Motivação

A utilização de metodologias e linguagens de análise orientadas a objetivos, como o *framework i**, o *framework* NFR ou a metodologia KAOS, tem como principal propósito a identificação das necessidades dos *stakeholders*[1] através de objetivos, e o seu posterior refinamento com a intenção de especificar os requisitos. Tal é feito durante a fase de elicitação de requisitos, que se encontra na fase inicial do desenvolvimento de um *software*.

Através da utilização das metodologias e linguagens referidas anteriormente, são criados modelos que pretendem elicitar, analisar e especificar sistemas de *software*, incluindo a especificação das diferentes partes envolvidas no sistema que se pretende modelar e construir, e de todas as suas dependências, que devem ser satisfeitas para que se alcancem os objetivos. Após a sua construção, esses modelos têm diversas utilizações, sendo as de maior importância a análise das suas propriedades e a comparação de alternativas para a construção do sistema de *software* e para a sua arquitetura.

Na maior parte das vezes, a construção desse tipo de modelos pretende representar sistemas de *software* de larga escala que, dada a sua natureza, têm uma grande complexidade associada. Existem dois tipos de complexidade que devem ser tidos em conta: essencial e acidental. A complexidade essencial é inerente ao problema que se pretende resolver e não pode ser evitada, sendo comum nos sistemas de *software* de larga escala.

[1] Parte interessada ou interveniente (pessoa, organização ou sistema) a quem o sistema de *software* interessa direta ou indiretamente. O sucesso do sistema depende da participação dos *stakeholders*.

Por sua vez, a complexidade acidental não é essencial ao problema que se pretende resolver [Bro87], e resulta da forma como os modelos são construídos. A complexidade pode contribuir para a existência de defeitos e problemas na gestão e na compreensão dos sistemas de *software* o que, por sua vez, leva a consequências nefastas. Algumas dessas consequências são o aparecimento de erros, incongruências e ambiguidades nas diversas fases do processo de desenvolvimento do sistema, o aumento dos custos de desenvolvimento, e uma grande insatisfação e descontentamento por parte dos *stakeholders*. Apesar de ser mais comum em sistemas de larga escala, devido à sua complexidade essencial, este género de problemas também pode ocorrer em sistemas de dimensão mais reduzida [Pre09], devido à complexidade acidental.

Existem alguns problemas que podem afetar a qualidade de modelos *i**. A criação destes modelos pode resultar em modelos com uma complexidade acidental elevada, que contribui para o crescimento do tamanho dos diagramas. Apesar de a sua complexidade poder ser alta, tal não significa que esses modelos estejam completos [Ale+06]. Adicionalmente, erros causados pelo mau uso da notação do *i** [WAC05] podem levar a um aumento dos custos de desenvolvimento do produto e da sua evolução [Pre09]. A qualidade dos modelos deve, então, ser uma preocupação constante, por forma a que sejam construídos modelos corretos, completos e cuja complexidade acidental seja minimizada, conseguindo aumentar-se a facilidade de compreensão dos mesmos, e diminuir os custos de desenvolvimento. Não obstante a sua importância, a gestão eficaz e efetiva da complexidade, completude e correção dos modelos de objetivos é, ainda, um problema em aberto.

1.3 Objetivos do Trabalho

O objetivo deste trabalho é encontrar uma forma de avaliar a qualidade nos modelos de requisitos *i**. Para esse efeito, propõe-se uma abordagem quantitativa para a avaliação da complexidade, da completude e da correção desses modelos, com vista à identificação de oportunidades para a sua melhoria.

A abordagem quantitativa proposta prende-se com a criação e validação de um conjunto de métricas, utilizadas para avaliar cada uma das componentes definidas anteriormente: a completude, a complexidade e a correção dos modelos *i**. As métricas são uma forma clara e precisa de avaliação no desenvolvimento de um sistema de *software*. Através da obtenção de métricas relativas à qualidade, é possível que sejam estabelecidos objetivos de melhoria no próprio processo de desenvolvimento, como forma de melhorar a sua qualidade. É possível, com uma avaliação quantitativa deste parâmetro, promover-se ajustes e alterações no processo de desenvolvimento, de forma a se conseguir reduzir ou eliminar causas de problemas que afetam, de forma significativa, o sistema de *software*.

As métricas propostas passam por uma avaliação experimental e por uma validação teórica, sendo aplicadas a um conjunto de casos de estudo reais e académicos.

1.4 Principais Contribuições

As principais contribuições desta dissertação prendem-se com a definição, recolha e avaliação de métricas para modelos de requisitos orientados a objetivos que utilizam o *framework i**, de maneira a dar suporte à avaliação quantitativa desses modelos. Esta avaliação permite melhorar os modelos, por forma a que estes se tornem mais compreensíveis e se diminuam os erros e ambiguidades, o que, consequentemente, faz com que o *software* tenha uma qualidade mais elevada. As métricas são propostas seguindo a abordagem *Goal-Question-Metric* (GQM) [BCR94; BCR14], de acordo com as boas práticas recomendadas pela *IEEE Computer Society* [IEE14], e são definidas quer informalmente, em língua natural, quer formalmente, através da linguagem OCL (do inglês *Object Constraint Language*) [Obj14a]. A definição informal pretende tornar mais compreensível a métrica a definir, enquanto que a correspondente definição formal evita eventuais ambiguidades que poderiam resultar em implementações inconsistentes das métricas propostas [Shu+02].

Uma outra contribuição é o desenvolvimento de uma ferramenta de criação e edição de modelos *i**, baseada na LDE i* [Nun09; Mon10], através da utilização de técnicas de construção de linguagens para domínio específico (em inglês *Domain Specific Language* - DSL [Fow11]). A ferramenta auxilia na criação de modelos *i**, permite verificar a boa formação desses modelos, torna automático o processo de recolha de métricas, e ajuda na avaliação dos modelos criados com base nas métricas propostas. Ao se obter modelos com melhor qualidade numa fase inicial do processo de desenvolvimento, essa qualidade é propagada para os artefactos das fases posteriores.

Recorre-se, ainda, a uma avaliação experimental, onde as métricas são aplicadas a um conjunto de casos de estudo reais e académicos. Esta avaliação visa validar as métricas propostas. Por fim, faz-se uma validação teórica das métricas, recorrendo-se a propriedades de Weyuker [Wey88].

As contribuições deste trabalho foram publicadas e apresentadas em *workshops* e conferências internacionais, nomeadamente no *International i* Workshop* 2013 [GGA13] e na conferência CAiSE'14 [GGA14].

1.5 Estrutura do Documento

Para além deste capítulo, o presente documento está organizado da seguinte forma:

- **Capítulo 2 – Enquadramento**: são apresentados alguns conceitos base referentes às áreas de engenharia de *software* e de engenharia de requisitos, assim como sobre engenharia de requisitos orientada a objetivos, metodologias que ajudam no processo de elicitação e avaliação de requisitos, e especificação formal de métricas.
- **Capítulo 3 – Trabalho Relacionado**: são apresentados diversos trabalhos que estão relacionados com esta dissertação, sendo feito o foco nas principais diferenças e/ou

contribuições. Os trabalhos avaliam ou preocupam-se com a qualidade de modelos de requisitos, assim como com a definição e especificação de métricas.

- **Capítulo 4 – Métricas para a Avaliação de Modelos *i**: é feita uma descrição do conjunto de métricas proposto, utilizando a abordagem GQM. É apresentado o meta-modelo que ajuda na formalização das métricas, as mesmas são descritas e definidas, e é apresentada a hierarquia completa da definição das métricas.

- **Capítulo 5 – Implementação do Protótipo**: são apresentadas as alternativas de implementação que foram consideradas, seguidas de quais as tecnologias utilizadas para a implementação da ferramenta que permite a recolha automática de métricas, e é fornecida informação sobre prevenção e deteção de erros por parte da ferramenta. É ainda apresentado como foi feito o suporte à interoperabilidade, bem como um cenário de utilização da ferramenta com a integração das métricas propostas.

- **Capítulo 6 – Avaliação**: são apresentados os casos de estudo aos quais as métricas são aplicadas. Os resultados são apresentados e descritos, sendo feita uma discussão sobre os mesmos. É, também, apresentada uma validação teórica das métricas, através de propriedades de Weyuker.

- **Capítulo 7 – Conclusões**: são apresentadas as conclusões desta dissertação, as suas limitações e o trabalho futuro.

2

Enquadramento

Este capítulo contém algumas bases referentes a engenharia de *software*, engenharia de requisitos, engenharia de requisitos orientada a objetivos, controlo de qualidade de *software*, e especificação formal de métricas. O seu principal objetivo é introduzir alguns conceitos importantes que estão relacionados com esta dissertação, fazendo um enquadramento da mesma.

2.1 Engenharia de *Software*

A engenharia de *software* é uma disciplina de engenharia que se preocupa com todos os aspetos da produção de sistemas de *software* [Som10]. O processo de *software*, nome dado à abordagem sistemática utilizada nesta área, inclui todas as atividades envolvidas no desenvolvimento de um programa, existindo 4 (quatro) atividades fundamentais que são comuns a todos os processos. Essas atividades são a especificação, o desenvolvimento (desenho e implementação), a validação e a evolução do *software*, podendo ser definidas da seguinte forma:

1. **Especificação do *software*** (identificação de requisitos) – os *stakeholders* e os engenheiros definem as funcionalidades do *software* a ser produzido e as restrições a que as suas operações estão sujeitas;

2. **Desenvolvimento do *software*** – o *software* é planeado de acordo com a fase anterior (desenho), e é implementado com o auxílio de linguagens de programação (implementação);

3. **Validação do *software*** – o *software* é analisado e testado por forma a garantir que

os requisitos foram cumpridos ou, mais concretamente, se o *software* corresponde ao que os *stakeholders* desejam;

4. **Evolução do *software*** – o *software* é corrigido por forma a serem removidos erros que apenas tenham sido detetados após o programa se tornar operacional. Também é nesta atividade que o *software* é modificado por forma a refletir mudanças nos requisitos dos *stakeholders* e do mercado.

Existem vários modelos que permitem representar, de forma abstrata, um processo de *software*. Exemplos desses modelos são o modelo em cascata [Roy70], o modelo em espiral [Boe86], o modelo de programação extrema (do inglês *extreme programming*) [Bec99], entre outros. Os modelos não são mutuamente exclusivos e são, muitas vezes, utilizados em conjunto. Tal é feito especialmente em sistemas de larga escala.

A engenharia de *software* pode ser dividida em 10 (dez) sub-disciplinas, sendo elas: requisitos, desenho, construção, teste, manutenção, gestão de configurações, gestão de engenharia, processos de engenharia, ferramentas e métodos de engenharia, e qualidade [Abr+04]. Dessas sub-disciplinas, as que merecem um estudo mais aprofundado no contexto desta dissertação são a engenharia de requisitos (subsecção 2.2) e qualidade de *software* (subsecção 2.4), uma vez que se pretende avaliar a qualidade de modelos de requisitos *i**.

2.2 Engenharia de Requisitos

A engenharia de requisitos está presente no início do ciclo de vida de um sistema de *software*. O seu principal objetivo prende-se com a identificação e análise de requisitos de um sistema, assim como com as restrições que este deve respeitar [Lam09]. Assim, os engenheiros de requisitos têm a necessidade de descobrir, perceber, formular, analisar e concordar em qual o problema a resolver, o porquê de ser necessário resolvê-lo, e quem tem a responsabilidade de o resolver. Desta definição podem-se retirar três perguntas fundamentais [Lam09]:

- **Porquê?** – que problema visa o sistema resolver (porque é que é necessário);
- **O quê?** – quais as características do sistema de *software*;
- **Quem?** – quem terá um papel ativo sobre o sistema de *sofware*.

Existem diversos processos que ajudam a responder às questões anteriores, sendo compostos por várias atividades ou fases do processo. Um desses processos pode ser visto como um processo em espiral, na medida em que suporta sucessivas iterações e incrementos, e passa pela execução das seguintes atividades [Lam09]:

- **Reconhecimento (compreensão) do domínio** – estudo do ambiente onde o *software* está inserido, assim como das suas características e problemas mais gerais. É necessário conhecer a estrutura e políticas organizacionais, assim como identificar e entrevistar os diversos *stakeholders*.

- **Elicitação de requisitos** – identificação de requisitos e pressupostos do sistema de *software*, partindo do estudo efetuado na atividade anterior. É uma fase crítica, uma vez que de um conjunto de requisitos pobre resulta um *software* pobre.

- **Avaliação e concordância** – realização de uma revisão independente dos materiais produzidos na fase anterior, tomando decisões informadas sobre problemas que sejam detetados. Prevê-se que, no final desta fase, tenham sido resolvidos todos os conflitos, e que tanto os *stakeholders* como os engenheiros estejam satisfeitos com o resultado final.

- **Especificação e documentação** – estruturação e documentação, em um documento de requisitos, das características resultantes das fases anteriores. Este documento pode ser dividido em vários, com partes específicas para cada um dos *stakeholders*.

- **Consolidação de requisitos** – correção de possíveis inconsistências, incongruências ou omissões antes de o documento de requisitos ser entregue à equipa de desenvolvimento. Esta fase pretende assegurar a qualidade do que foi especificado anteriormente.

Existe um outro processo de engenharia de requisitos semelhante ao anterior, embora adotando uma cobertura e subdivisão em atividades diversas, bem como uma terminologia ligeiramente diferente. Trata-se do processo presente em [KS04]. A correspondência entre esses dois processos pode ser vista na tabela 2.1.

Tabela 2.1: Correspondência entre os processos presentes em [Lam09] e em [KS04]

Atividades em [Lam09]	**Atividades em [KS04]**
Reconhecimento do domínio	Elicitação de requisitos
Elicitação de requisitos	Elicitação de requisitos
Avaliação e concordância	Análise e negociação de requisitos
Especificação e documentação	Documentação de requisitos
Consolidação de requisitos	Validação de requisitos
—	Gestão de requisitos

Para além da junção do reconhecimento do domínio e da elicitação de requisitos em uma só atividade com a designação de elicitação de requisitos, [KS04] adiciona a atividade gestão de requisitos. Os objetivos dessa atividade prendem-se com a gestão de mudanças nos requisitos acordados, a gestão do relacionamento entre requisitos, e a gestão

de dependências entre o documento de requisitos e outros documentos eventualmente produzidos durante todo o processo. Esta atividade torna-se importante uma vez que podem existir alterações nos requisitos, ou até mesmo haver a necessidade de incluir requisitos novos. Um outro processo, que também tem em conta este aspeto, é a abordagem apresentada em [NE00], onde a gestão é designada como evolução de requisitos.

2.2.1 Classificação de Requisitos

Os requisitos podem ser definidos como propriedades que devem ser especificadas por forma a resolver algum problema do mundo real. São declarações que identificam recursos ou funções necessárias a um sistema de *software* para este satisfazer as necessidades dos clientes. Desta forma, um requisito pode ser um atributo, uma característica ou uma qualidade que o *software* possui. Um requisito informa o que o sistema de *software* deve fazer, mas não como o fazer [BD09]. Os requisitos de um sistema de *software* podem ser classificados em 3 (três) categorias: requisitos funcionais, não funcionais, ou de domínio [Som10].

Os **requisitos funcionais** descrevem os serviços que um sistema deve disponibilizar, como deve reagir a certos dados de entrada e como se deve comportar em determinadas situações. Em alguns casos, também podem explicitar o que o sistema não deve fazer. Descrevem a funcionalidade ou serviços do sistema, e dependem do tipo de *software*, dos utilizadores previstos, e do tipo de sistema onde o *software* vai ser utilizado. Exemplo de um requisito funcional, no contexto de um sistema bancário: *"Cada pedido de levantamento de dinheiro deve ter um identificador único (`withdraw_id`)"*.

Os **requisitos não funcionais** descrevem atributos de qualidade (propriedades do sistema, como fiabilidade e tempo de resposta) e as restrições de um sistema, tais como restrições temporais, restrições no processo de desenvolvimento, padrões, entre outros. Explicam como o sistema de *software* deve ser desenvolvido, indicando a forma como este satisfaz os requisitos funcionais e apresentando restrições às quais o sistema deve obedecer. Estes requisitos também incluem as linguagens de programação e o método de desenvolvimento a serem utilizados. Exemplo de um requisito não funcional: *"O sistema da caixa multibanco deve ejetar automaticamente o cartão se o cliente não digitar o código pin em um intervalo de tempo superior a 4 segundos"*. Estes requisitos podem, ainda, ser classificados em três categorias: de produto, organizacionais e externos. Os **requisitos de produto** especificam que o produto se deve comportar de determinada maneira, tal como a sua rapidez e a sua fiabilidade, por exemplo. Os **requisitos organizacionais** são consequência de políticas e procedimentos organizacionais, tal como os padrões de processo utilizados. Os **requisitos externos** são requisitos que surgem a partir de fatores externos ao sistema, tais como requisitos legislativos e éticos.

Por fim, os **requisitos de domínio** são derivados do domínio da aplicação do sistema e descrevem as características desse mesmo domínio. Podem ser requisitos funcionais ou restrições. Um exemplo são os requisitos de domínio de uma linha de produtos de

telemóvel, em que os requisitos expressam a variabilidade dos vários tipos de telemóvel. Este tipo de requisitos apresenta alguns problemas, nomeadamente de compreensão. Os requisitos são expressos na linguagem do domínio da aplicação, o que nem sempre é bem compreendido pelos engenheiros de *software*. Também existe o problema dos requisitos implícitos. Os especialistas do domínio compreendem a área tão bem que, por vezes, não se apercebem da necessidade de tornar os requisitos do domínio explícitos para quem não faz parte da área.

2.2.2 Abordagens de Requisitos

A elicitação de requisitos pode apresentar alguns problemas que, se não forem resolvidos, irão prejudicar as fases seguintes do processo de desenvolvimento do *software*. Alguns desses problemas são o facto de os requisitos poderem não refletir as necessidades reais dos utilizadores do sistema; não serem definidos de forma precisa, ou seja, serem inconsistentes, incompletos, ou mesmo omissos; serem ambíguos e a sua especificação ser potenciadora de interpretações inconsistentes entre os diferentes *stakeholders*; a sua alteração ter um custo elevado depois de estarem acordados e começarem a ser implementados; a negligência de requisitos não funcionais, por não serem tão óbvios como os funcionais; entre outros. Estes problemas podem ser divididos em 3 (três) categorias: âmbito, compreensão e volatilidade:

- **Problemas de âmbito** – os limites do sistema de *software* são mal definidos e/ou a sua especificação é confusa;
- **Problemas de compreensão** – os *stakeholders* nem sempre sabem o que realmente desejam e/ou não se exprimem de forma clara e inequívoca;
- **Problemas de volatilidade** – os requisitos não são estáticos, podem ser alterados ao longo do tempo.

Por forma a prevenir erros e ultrapassar alguns dos problemas explicitados anteriormente, têm vindo a ser desenvolvidas técnicas ou abordagens que têm como objetivo melhorar a comunicação entre as diferentes partes envolvidas. Essas técnicas são utilizadas para a descrição, estruturação e apresentação de requisitos. Diferentes abordagens combinam características de mais do que um destes paradigmas sendo, desse modo, abordagens híbridas. Segue-se uma pequena introdução às mais relevantes no contexto desta dissertação.

- **Engenharia de Requisitos Orientada a Objetivos** [Lam01] – utiliza a noção de objetivo para elicitar, elaborar, estruturar, especificar, analisar, negociar, documentar e modificar requisitos de um sistema. Os objetivos são declarações que exprimem propriedades que o sistema de *software* deve garantir. Podem ser formulados em diferentes níveis de abstração, tornando a explicitação dos requisitos mais adequada

aos diversos *stakeholders*. Um exemplo de notação que utiliza este tipo de abordagem é o *framework i**.

- **Engenharia de Requisitos Orientada a Objetos** [Kas03; Boo+07] – baseia-se no encapsulamento da informação sobre o produto, o processo, e a sua funcionalidade. Os requisitos do sistema de *software* são traduzidos através de objetos, que contém informação sobre a sua funcionalidade, o seu comportamento e as suas interações com os outros objetos. Um exemplo de uma notação que utiliza este tipo de abordagem é a UML (do inglês *Unified Modeling Language*) [Obj14b].

- **Engenharia de Requisitos Orientada a Agentes** [Yu01] – é combinada com a engenharia de requisitos orientada a objetivos, e utiliza o agente como principal abstração, permitindo modelar sistemas complexos através de uma metáfora social. Permite capturar características como requisitos não funcionais e organizacionais de forma explícita. Um exemplo de uma notação que utiliza este tipo de abordagem é o *framework i**.

- **Engenharia de Requisitos Orientada a Pontos de Vista** [Som+97; KS04] – tem como propósito identificar e organizar os requisitos de acordo com diferentes pontos de vista (do inglês *viewpoint*). Um *viewpoint* representa um conjunto de informações sobre o sistema, do ponto de vista de um determinado *stakeholder*. Ao recolher pontos de vista de diferentes fontes, torna-se mais fácil compreender os possíveis conflitos entre *stakeholders*. Um exemplo de uma notação que utiliza este tipo de abordagem é o CORE (do inglês *COntrolled Requirement Expression*) [Mul79] e PREview (do inglês *Process and Requirements Viewpoints*) [SSV96].

- **Engenharia de Requisitos Baseada em Cenários** [Som10] – utiliza exemplos de comportamentos existentes, os cenários, para completar a descrição do sistema de *software* a desenvolver. Torna mais fácil a compreensão por parte dos *stakeholders*, uma vez que utiliza exemplos reais. Um exemplo de utilização desta técnica é em conjunto com UML, nomeadamente quando se pretende definir cenários para especificar casos de uso. Um caso de uso pode ser visto como um tipo cujas instâncias são cenários.

- **Engenharia de Requisitos Orientada a Aspetos** [Ara+05] – pretende identificar e especificar propriedades transversais (do inglês *crosscutting concerns*) ou aspetos. Exemplos dessas propriedades são a disponibilidade e a segurança. Desta forma é possível identificar as influências dessas propriedades noutros requisitos do sistema de *software*, reforçando a modularidade desse mesmo sistema através de mecanismos de abstração, representação e composição adequados para o domínio de engenharia de requisitos.

Das abordagens anteriores, a que merece um estudo mais aprofundado, no contexto desta dissertação, é a engenharia de requisitos orientada a objetivos (subsecção 2.3).

2.3 Engenharia de Requisitos Orientada a Objetivos

Na engenharia de requisitos orientada a objetivos, os requisitos são apresentados como objetivos que o sistema de *software* deve ser capaz de satisfazer. Para tal, os requisitos são identificados, analisados, negociados, documentados e atribuídos a componentes do sistema de *software* ou a agentes do ambiente [Lam01]. Assim, os requisitos definem a forma como os objetivos podem ser atingidos, bem como as diferentes alternativas para os alcançar. Os objetivos são declarações que podem ser formuladas em diferentes níveis de abstração, indo desde um nível mais geral, no qual são designadas preocupações estratégicas, até um nível mais específico, sendo consideradas como preocupações técnicas. Estes diferentes níveis de abstração permitem que a explicação dos requisitos aos vários *stakeholders* seja feita de forma mais adequada para cada um deles. As principais motivações da engenharia de requisitos orientada a objetivos são descritas a seguir [Lam01]:

- Devido aos vários níveis de abstração obtidos pelo refinamento dos objetivos, oferece um quadro global para documentar os requisitos, assim como auxilia na comunicação com os *stakeholders*. Os grafos de refinamento de objetivos, por exemplo, permitem apresentar ligações de rastreabilidade entre níveis mais abstratos (objetivos estratégicos) e níveis mais concretos (requisitos técnicos, podendo facilitar a compreensão por parte dos *stakeholders*).

- Através do refinamento alternativo dos objetivos e da atribuição de responsabilidades alternativas, é possível explorar várias configurações do sistema. Os refinamentos alternativos auxiliam os engenheiros de requisitos nas tomadas de decisão, permitindo a avaliação das opções que foram tomadas, assim como explorar outras opções.

- Evitam requisitos irrelevantes, fornecendo um critério preciso para a pertinência dos requisitos: um requisito é pertinente para um conjunto de objetivos, em um dado domínio, se a sua especificação for utilizada para atingir pelo menos um objetivo.

- A formalização dos objetivos permite provar a completude e integridade do sistema: a especificação está completa quando todo o conjunto de objetivos consegue ser atingido a partir da especificação e das propriedades conhecidas sobre o domínio em questão.

- Os objetivos podem ser utilizados como suporte à deteção, gestão e resolução de conflitos entre requisitos.

Das metodologias que suportam engenharia de requisitos orientada a objetivos, serão apresentadas o *framework i** (subsecção 2.3.1), o *framework NFR* (subsecção 2.3.2) e a metodologia KAOS (subsecção 2.3.3). Por ser a que merece um estudo mais aprofundado no

contexto desta dissertação, o *framework i** será a metodologia mais detalhada, enquanto que as outras 2 (duas) serão apenas alvo de uma breve introdução.

2.3.1 *Framework i**

O *framework i** [Yu95; Yu97] é uma abordagem orientada a agentes e a objetivos, centrando-se nos *stakeholders* do sistema e nas suas relações. É uma linguagem de modelação apropriada para modelar sistemas na fase de elicitação de requisitos, por forma a se melhor compreender o domínio do problema. Tem como propósito a modelação de ambientes organizacionais, oferecendo uma representação formal das relações estratégicas entre agentes e os seus comportamentos. Assim sendo, o objetivo desta metodologia é analisar a forma como os objetivos dos diferentes atores podem ser alcançados, tendo em conta as relações entre atores externos e o sistema. Consiste em dois modelos básicos:

- **Modelo de Dependência Estratégica** (SD, do inglês *Strategic Dependency*) – descreve as relações de dependência entre os atores do sistema. Tem como objetivo captar a intenção dos processos dentro da organização e a importância que cada um desses processos tem em relação aos participantes, garantindo a abstração quanto aos restantes detalhes do sistema;

- **Modelo de Razão Estratégica** (SR, do inglês *Strategic Rationale*) – explica como os atores atingem os seus objetivos. É usado para explorar a lógica por detrás dos processos do sistema, descrevendo os seus interesses e preocupações. É um modelo complementar ao anterior, especificando, para além dos atores e das suas dependências, os objetivos funcionais e não funcionais, as tarefas, os recursos e as crenças, utilizando as fronteiras de cada ator para especificar os seus assuntos internos.

Os elementos presentes nesta metodologia podem ser classificados em 2 (duas) categorias, atores e nós, e são os seguintes:

- **Atores** – entidades ativas, que tanto podem ser seres humanos como sistemas, capazes de realizar ações independentes. Os atores podem-se relacionar entre si através de ligações de associação e/ou de dependência. A noção de ator pode ser expandida para tipos mais específicos: agentes, posições e papéis:

  - **Agentes** – representam uma identidade real, como um ser humano, uma organização ou um sistema de *software*/*hardware*;
  - **Posições** – representam uma coleção de papéis, no contexto de uma organização, sendo uma abstração intermediária usada entre um papel e um agente;
  - **Papéis** – caracterização abstrata do comportamento de um ator em um determinado contexto. Representa uma responsabilidade que um agente concreto possui por desempenhar um determinado papel.

- **Objetivos** – estão associados a atores e são requisitos que devem ser cumpridos;
- ***Softgoals*** – representam objetivos que não têm um critério bem definido para determinar se estão, ou não, a ser satisfeitos;
- **Tarefas** – indicações sobre algo que deve ser feito pelos atores e podem ser vistas como uma solução (operações e processos) no sistema que se pretende desenvolver;
- **Recursos** – entidades, físicas ou não, que podem ser disponibilizadas pelo sistema;
- **Crenças** (*Belief*) – condições sobre o mundo que o ator pensa serem verdadeiras.

A figura 2.1 ilustra a notação básica do *framework i**, mostrando os diferentes tipos de atores e nós, assim como a fronteira de um ator.

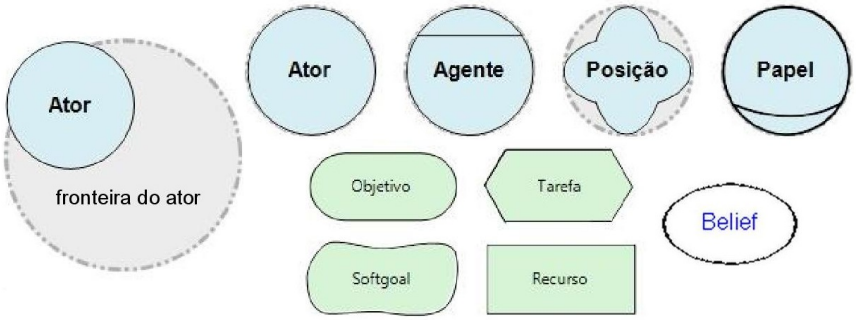


Figura 2.1: Notação básica do *framework i** (tipos de atores e elementos)

A nível de relações entre os diferentes elementos, existem os seguintes tipos:

- **Associações** – permitem descrever graficamente as relações entre atores, estando subdivididas em:
  - ***ISA*** – representa uma generalização, com um ator a ser um caso especializado de outro ator;
  - ***Covers*** – descreve uma relação entre uma posição e um papel;
  - ***Is part of*** – permite representar sub-partes de agentes, posições e papéis;
  - ***Occupies*** – descreve uma relação entre um agente e uma posição;
  - ***Plays*** – descreve uma relação entre um agente e um papel;
  - ***INS*** – representa uma instanciação de um agente noutro agente.
- **Ligações *Means-end*** – utilizadas para ligar uma tarefa a um objetivo, indicando um meio (tarefa) para atingir um fim (objetivo);
- **Ligações de Decomposição** – permitem especificar uma tarefa, indicando sub-tarefas, sub-objetivos, recursos e *softgoals* necessários para a satisfazer;

- **Ligações de Contribuição** – permitem indicar como é que um elemento contribui para alcançar os atributos de qualidade especificados pelos *softgoals*;
  - ***Make*** – contribuição positiva com força suficiente para satisfazer um *softgoal*;
  - ***Break*** – contribuição negativa com força suficiente para negar um *softgoal*;
  - ***Some+*** – contribuição positiva cuja força é desconhecida;
  - ***Some-*** – contribuição negativa cuja força é desconhecida;
  - ***Help*** – contribuição parcialmente positiva, não suficiente por si só para satisfazer o *softgoal*;
  - ***Hurt*** – contribuição parcialmente negativa, não suficiente por si só para negar o *softgoal*;
  - ***Unknown*** – contribuição cuja polaridade é desconhecida;
  - ***And*** – o ascendente é satisfeito se todos os descendentes o forem;
  - ***Or*** – o descendente é satisfeito se qualquer um dos descendentes o for;
- **Ligações de Dependência** – indicam que um ator (*depender*) depende de outro ator (*dependee*) para realizar algo (*dependum*). O *dependum* pode ser um objetivo, um *softgoal*, uma tarefa, ou um recurso. As ligações podem ser dos seguintes tipos:
  - ***Open*** – a não obtenção do *dependum* afeta o *depender*, mas não criticamente;
  - ***Committed*** – a não obtenção do *dependum* faz com que alguma ação necessária para atingir o objetivo falhe no *depender*;
  - ***Critical*** – a não obtenção do *dependum* faz com que todas as ações falhem.

As figuras 2.2 e 2.3 ilustram as diferentes relações permitidas, entre atores e elementos, no *framework i**. Como se pode observar, as únicas relações permitidas no modelo SD são as associações (figuras à esquerda) e as dependências (figura à direita). No modelo SR, acrescentam-se as ligações de decomposição, as ligações de contribuição e a ligação de *means-end*.

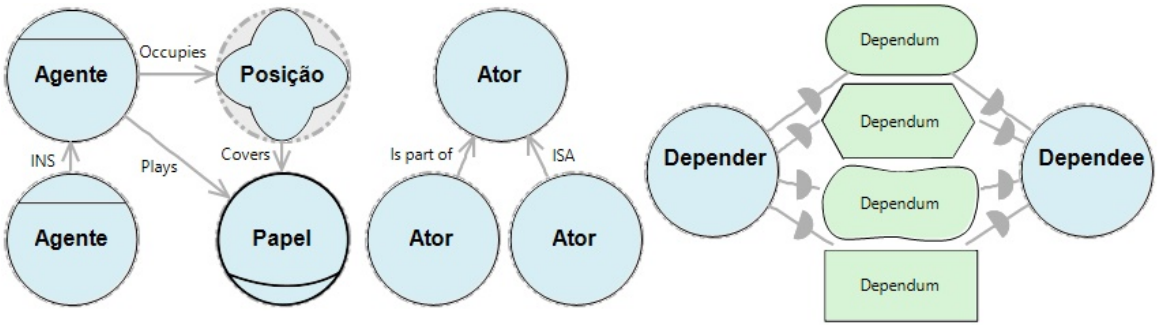


Figura 2.2: Notação das relações permitidas no modelo SD do *framework i**

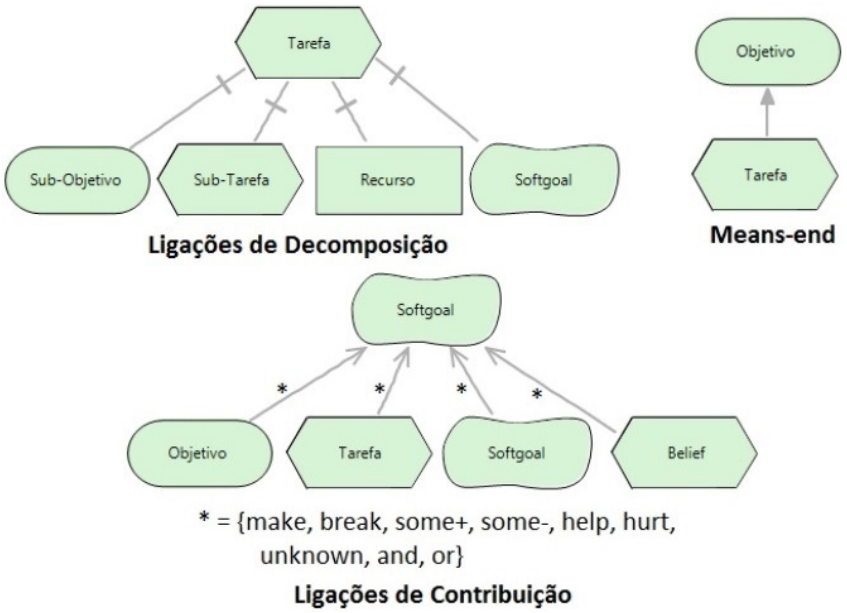
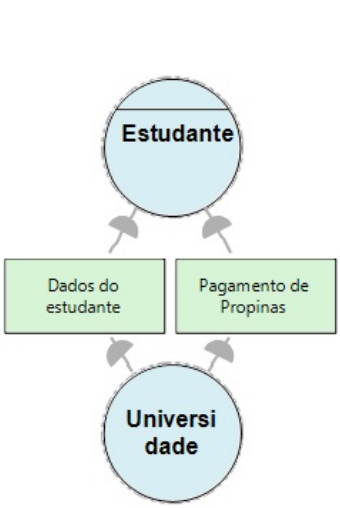
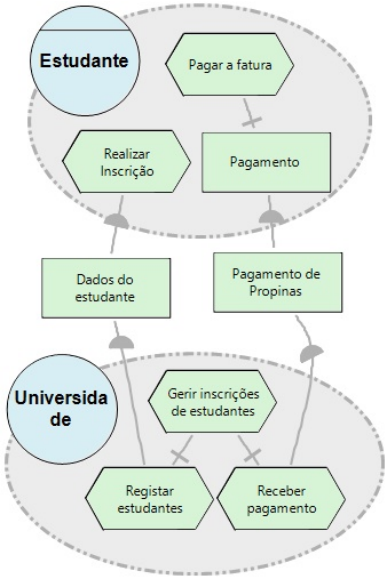


Figura 2.3: Notação das relações permitidas no modelo SR do *framework i**

Por forma a se compreender melhor os conceitos apresentados, a figura 2.4 demonstra modelos SD e SR simples de uma inscrição na universidade. No modelo SD, o ator **Universidade** está dependente do ator **Estudante** para receber os recursos **Dados do Estudante** e **Pagamento de Propinas**, não sendo fornecida informação sobre os assuntos internos aos atores. No modelo SR, os atores são detalhados. Assim, o ator **Estudante** é responsável pelas tarefas **Realizar inscrição** e **Pagar a fatura**, e por fornecer o recurso **Pagamento**. O ator **Universidade** é responsável pela tarefa **Gerir inscrições de estudantes**, que pode ser decomposta nas tarefas **Registar estudantes** e **Receber pagamento**.



(a) Modelo SD



(b) Modelo SR

Figura 2.4: Modelos SD e SR, adaptados de [wik14e]

Existem diversas ferramentas que permitem construir modelos seguindo a notação do *framework i**, como OpenOME [HYY11; Ope14], jUCMNav [RKA06; jUC14], DesCARTES [Kol+06; Des14], e LDE i* [Nun09; Mon10]. Uma lista mais completa e detalhada pode ser vista em [wik14d].

2.3.1.1 Variantes do *Framework i**

A versão original do *framework i**, utilizada nesta dissertação, é designada por Yu'95, e apresentada na página *wiki* do *i** [wik14e], um dos mais importantes locais com informação oficial sobre este *framework*. No entanto, existem algumas variantes à versão original, sendo que 2 (duas) delas serão brevemente descritas em seguida, uma vez que são referidas ao longo desta dissertação: Tropos e GRL.

A Tropos [GMS05; Tro14b] é uma metodologia de desenvolvimento de *software*, suportando todas as fases do desenvolvimento. São adotados os conceitos da variante Yu'95, tais como atores, elementos e relações. Conta ainda com variantes dentro da própria metodologia, tal como Secure Tropos, Iterative Tropos, entre outros.

A GRL (do inglês *Goal-oriented Requirements Language*) [LY04] é uma linguagem de modelação para análise de requisitos, orientada a objetivos e a agentes. Esta abordagem encontra-se centrada nos requisitos não funcionais e é composta por 3 (três) conceitos principais: elementos intencionais, relações intencionais e atores. Adicionalmente, pode ser representada de 3 (três) formas distintas: representação gráfica dos modelos, representação textual dos modelos, e representação dos modelos em XML (do inglês *eXtensible Markup Language*) [W3C14].

2.3.1.2 Comparação Sistemática de Ferramentas

Como visto anteriormente, existem diversas ferramentas disponíveis para a construção de modelos *i**, sendo que cada uma delas possui diferentes funcionalidades e propósitos específicos. Tendo em conta que a presente dissertação propõe a criação de uma ferramenta, com recurso a linguagens para domínio específico, que permita a criação deste tipo de modelos, torna-se importante estudar as ferramentas existentes, por forma a avaliar como suportam a correção de modelos, assim como identificar aspetos positivos, aspetos que podem ser melhorados, e aspetos que não estão a ser tidos em conta.

Realizou-se um estudo inicial [GGA13] das ferramentas presentes na página *wiki* do *i** [wik14d]. A página disponibiliza uma comparação [wik14a] entre as diferentes ferramentas, cobrindo informação sobre **qual o propósito** da ferramenta; detalhes práticos relacionados com **disponibilidade**, **plataforma** base, e nível de **maturidade**; assim como detalhes sobre a **adequação da modelação**, **usabilidade**, **extensibilidade** e **interoperabilidade** da ferramenta. No entanto, esta comparação não cobre as características sintáticas e semânticas suportadas pelas diferentes ferramentas. Utilizou-se, então, a descrição do *framework i**, presente na página *wiki*, como base para a comparação entre ferramentas, procurando-se saber: i) quais das construções sintáticas descritas na *wiki* são suportadas

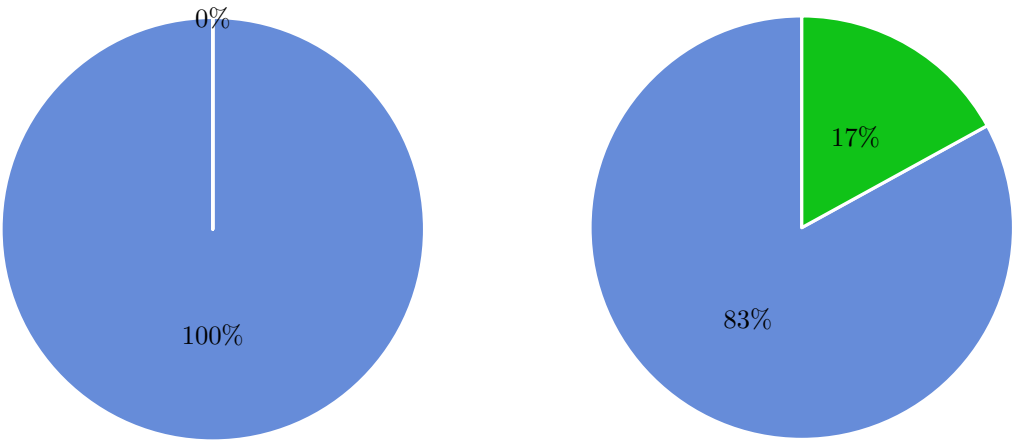
pela ferramenta, e ii) até que ponto cada ferramenta suporta verificação de erros presentes nos modelos construídos com o seu auxílio.

O critério para a inclusão de determinada ferramenta na análise foi a sua presença na página *wiki* do *i** e a existência de um URL funcional. Algumas ferramentas listadas não estavam disponíveis no URL indicado, pelo que foram excluídas da análise. A tabela 2.2 apresenta as ferramentas analisadas.

Tabela 2.2: Ferramentas analisadas

Ferramenta	**Instituição**	**Variante *i****	**Tecnologia**
OpenOME [Ope14]	Univ. Toronto	Yu'95	Java (JRE)
TAOM4E [TAO14]	Univ. Trento	Tropos	Eclipse *plug-in*
GR-Tool [Tro14a]	Univ. Trento	Tropos	Java (JRE)
STS-Tool [STS14]	Univ. Trento	Tropos	Java (JRE)
jUCMNav [jUC14]	Univ. Ottawa	GRL	Eclipse *plug-in*
DesCARTES [Des14]	U.C. Louvain	Yu'95/Tropos	Eclipse *plug-in*

A análise da cobertura sintática pretende determinar se a ferramenta apresenta: a) a sintaxe básica do *i**, e b) a notação gráfica da sintaxe do *i**, de acordo com a apresentada na página *wiki*. Como se pode observar pela figura 2.5, verificou-se que nenhuma das ferramentas possui todos os elementos presentes no *framework i**, e que 83% delas tem pelo menos um elemento cuja representação gráfica difere da apresentada na página *wiki*. Em termos mais rigorosos, a sintaxe concreta suportada pela maioria das ferramentas não coincide com a sintaxe oficial do *i**, apresentada na página *wiki*.

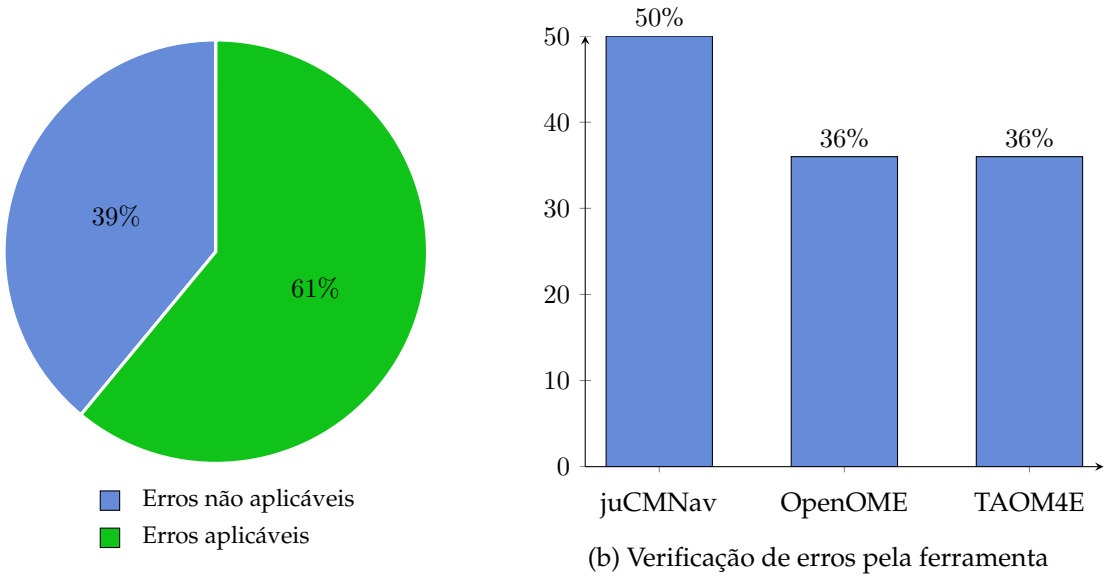


(a) Sintaxe básica do *i** (b) Notação gráfica do *i**

Figura 2.5: Resultados da análise da cobertura sintática

A análise da verificação de erros pretende determinar o nível de verificação de correção dos modelos criados com a ferramenta. Utilizando as descrições e guias presentes na página *wiki* do *i**, criou-se um catálogo de problemas típicos, e analisou-se se as ferramentas verificam quando é cometido um erro de modelação.

As várias ferramentas suportam prevenção e/ou correção de erros. A abordagem de prevenção impede o utilizador de cometer determinados erros do ponto de vista da modelação, não permitindo que este realize certas ações. A abordagem de correção permite que o utilizador realize a ação que deseja, mas informa-o de que o modelo não está correto. Após a análise das ferramentas, e como se pode observar na figura 2.6, conclui-se que, em geral, o nível de verificação de erros nos modelos criados não é muito elevado. Em média, cerca de 39% dos erros considerados não são aplicáveis, uma vez que as ferramentas em questão não suportam os elementos *i** necessários. A juCMNav é a ferramenta com o número mais elevado de verificação de erros, com uma percentagem de verificação de 50%, seguida pelas ferramentas OpenOME e TAOM4E (ambas com cerca de 36%). De notar que a percentagem de verificação de erros tem em conta os erros aplicáveis, ou seja, tem em conta o fragmento da linguagem suportado pela ferramenta em questão.



(a) Erros considerados no catálogo

(b) Verificação de erros pela ferramenta

Figura 2.6: Resultados da análise da verificação de erros

Do estudo realizado, pode-se concluir que nenhuma das ferramentas existentes suporta todos os elementos do *framework i**. Da mesma forma, a maior parte dos erros cometidos pelos utilizadores não são verificados pelas ferramentas, o que potencia a construção de modelos incorretos. Torna-se, então, importante a construção de uma ferramenta que disponibilize todos os elementos presentes no *framework i**, e que permita verificar a correção dos modelos criados pelos utilizadores. Esta verificação da correção passa pela prevenção de alguns erros, evitando-os por construção, assim como pela deteção

de outros. A prevenção é feita em tempo real, não permitindo que o utilizador realize determinadas ações incorretas. No entanto, enquanto se está a modelar, podem existir alguns erros devido ao facto de ainda não terem sido modelados todos os elementos e relações necessários. Um exemplo de um erro deste tipo é a existência de atores sem ligações. Neste caso, não se deve impedir a colocação de um ator no modelo, mas o erro deve ser detetado quando o utilizador pedir a verificação da correção do modelo, carregando numa determinada opção da ferramenta. Assim, a deteção dos erros é feita posteriormente, a pedido do utilizador, e mostra tanto o erro, como uma pequena descrição do mesmo, e sugestões sobre como o corrigir.

2.3.2 *Framework NFR*

O *framework* NFR [TC99] é uma abordagem orientada ao processo, que oferece técnicas para justificar decisões de desenho durante o processo de desenvolvimento de um *software*. Adicionalmente, ajuda a compreender o impacto que essas decisões terão nos requisitos não funcionais. Trata os requisitos não funcionais como objetivos a serem atingidos, e utiliza a noção de *softgoal*. Os *softgoals* representam objetivos que não têm um critério bem definido para determinar se estão ou não a ser satisfeitos.

Esta abordagem é baseada em grafos SIG (grafos de interdependência de *softgoals*, do inglês *Softgoal Interdependency Graph*) que, como o nome indica, é um grafo que representa *softgoals* e as suas interdependências. Um grafo é composto por nós e relações entre eles. Os nós são designados por *softgoals*, e podem ser de 3 (três) tipos:

- ***Softgoals* NFR** – requisitos não funcionais de alto nível que devem ser satisfeitos;
- ***Softgoals* de operacionalização** – modelam técnicas para satisfazer os *softgoals* NFR, ou seja, são possíveis soluções ou alternativas de desenho (operações, processos, representações de dados, entre outros) que ajudam a atingir um *softgoal* NFR;
- ***Softgoals* de argumentação** – pretendem justificar a lógica de desenho relativamente a *softgoals* e ligações de interdependência, explicando o seu contexto.

As relações entre nós podem ser de 2 (dois) tipos diferentes, sendo eles:

- **Interdependências** – conjunto de relacionamentos entre os vários *softgoals*. Podem ser explícitas ou implícitas;
- **Contribuições** – conjunto de regras que permitem inferir interações entre *softgoals*. Podem ser negativas, parcialmente negativas, parcialmente positivas ou positivas.

O refinamento dos *softgoals* pode ser feito através de *and-refinement* e *or-refinement*, permitindo decompor os objetivos e reconhecer várias formas alternativas de os garantir.

Existe, ainda, um processo de avaliação dos *softgoals*, que mede o impacto que a importância de cada objetivo tem em relação aos objetivos de alto nível. Esta avaliação pode ter os seguintes valores: *weakly denied*, *denied*, *conflict*, *undecided*, *weakly satisfied* e *satisfied*.

A figura 2.7 pretende ilustrar os principais componentes do *framework* NFR, descritos anteriormente.

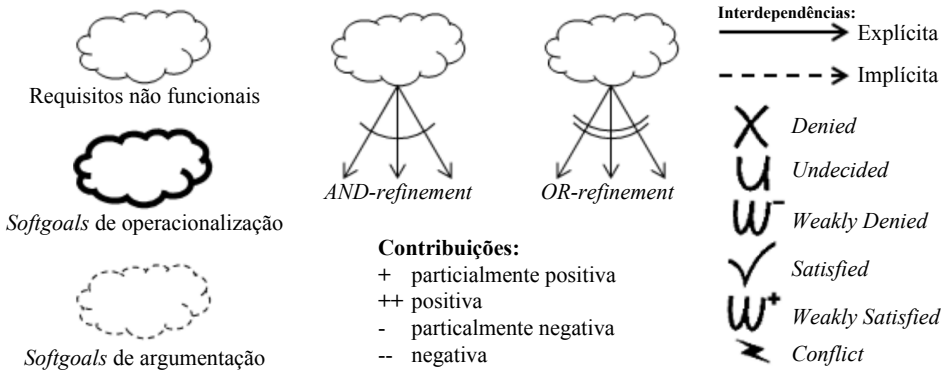


Figura 2.7: Componentes do *framework* NFR

A ferramenta OME (do inglês *Organization Modelling Environment*) [YY14] permite construir modelos seguindo o *framework* NFR.

2.3.3 KAOS

A metodologia KAOS [DLF93; Lam14] pretende dar suporte ao processo de elaboração e aquisição de requisitos, baseando-se na decomposição e refinamento de objetivos. É uma abordagem sistemática que permite descobrir e estruturar requisitos enquanto evita requisitos ambíguos ou irrelevantes, permite comunicação eficiente e fácil entre *stakeholders*, e clarifica as responsabilidades dos mesmos. Adicionalmente, também oferece mecanismos para escolher entre diferentes alternativas, gerir conflitos, e refinar objetivos por forma a estruturar requisitos complexos. É composta por 4 (quatro) tipos de modelos: de objetivos, de responsabilidade, de objeto e de operação. Utiliza os seguintes conceitos principais:

- **Objetivos** – metas que o sistema deve ser capaz de cumprir. Podem ser decompostos e descritos por objetivos de baixo nível, através de *and-refinement* e *or-refinement*;
- **Agentes** – seres humanos (agentes do ambiente) ou componentes automatizados (agentes do sistema) responsáveis por atingir expetativas dos requisitos, atuando sobre determinadas operações ou objetivos;
- **Expectativas** – requisitos de agentes do ambiente. São utilizadas para demonstrar como é que o sistema de *software* e o seu ambiente têm que cooperar por forma a atingir os objetivos;
- **Requisitos** – tipo de objetivo de baixo nível, a ser atingido por um agente de sistema;
- **Objetos** – elementos de interesse no sistema de *software*, caracterizados por um conjunto de atributos que definem diferentes estados nos quais o objeto pode transitar;

- **Operações** – permitem a transição dos objetos através de relações de entrada e saída. São caracterizadas por pré-condições, pós-condições e condições de desencadeamento;
- **Obstáculos** – abordagem defensiva, que impede que os objetivos sejam atingidos. Permitem identificar inconsistências dentro do domínio do sistema de *software*, podendo ser resolvidos através de ligações de resolução;
- **Conflitos** – existem quando não é possível satisfazer 2 (dois) objetivos simultaneamente. Permitem identificar inconsistências dentro do domínio do sistema de *software*.

Existem 2 (dois) critérios que permitem verificar a completude do modelo de objetivos, sendo eles:

1. Um modelo de objetivos é dito completo no que diz respeito à relação de refinamento se e só se cada objetivo folha é uma expetativa, uma propriedade do domínio ou um requisito;
2. Um modelo de objetivos é dito completo no que diz respeito à relação de responsabilidade se e só se cada requisito é colocado sob a responsabilidade de um e um só agente, ou implicitamente se o requisito refina outro que tenha sido colocado sob a responsabilidade de algum agente.

A figura 2.8 pretende ilustrar os diferentes modelos e os diversos componentes da metodologia KAOS, assim como ajudar a compreender a relação entre os principais conceitos e definições.

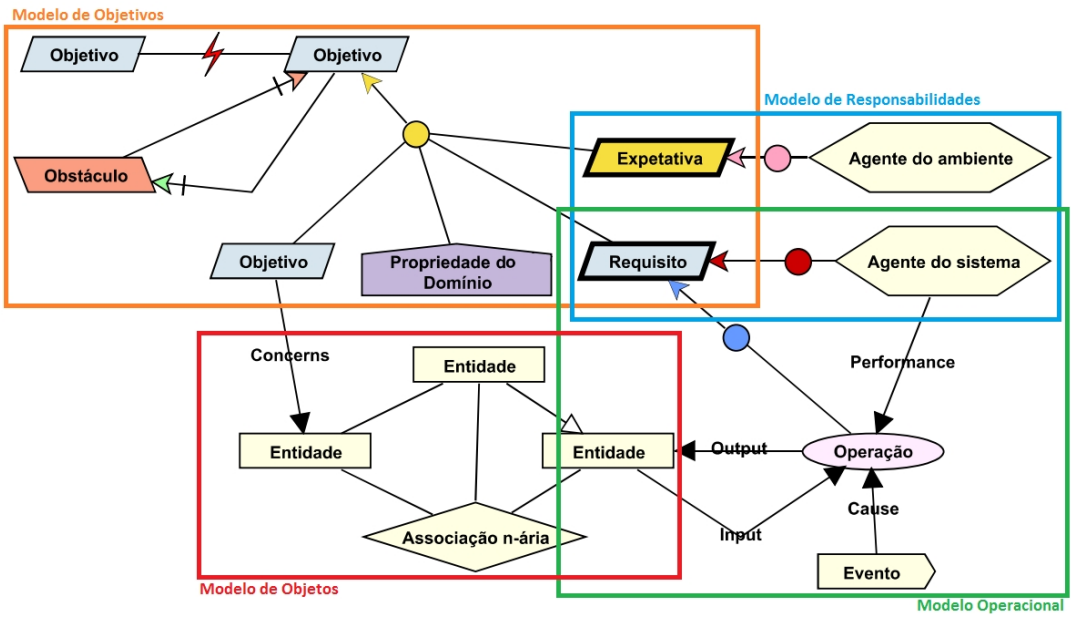


Figura 2.8: Modelos da abordagem KAOS, adaptado de [Res14a]

A ferramenta Objectiver [Res14b] é uma ferramenta comercial de modelação que permite construir modelos seguindo a metodologia KAOS. A ferramenta modularKAOS é uma ferramenta experimental, desenvolvida no âmbito de uma dissertação de mestrado [Dia09], e posteriormente aperfeiçoada [Mon10; Esp12]. Trata-se de um editor de modelos KAOS baseado em linguagens para domínios específicos.

2.4 Controlo de Qualidade de *Software*

Por forma a garantir a qualidade dos sistemas de *software* construídos, têm sido propostas várias alternativas, tais como medições, *benchmarking*, validações do sistema, entre outras. No contexto desta dissertação, a técnica que merece maior atenção é a *Goal-Question-Metric*, que será explicada em seguida.

A abordagem *Goal-Question-Metric* (GQM) [BCR94; BCR14] é uma técnica de definição de métricas recomendada pela *IEEE Computer Society* [IEE14], sendo utilizada como padrão pela comunidade de engenharia de *software* experimental. É uma abordagem orientada a objetivos para medições de sistemas de desenvolvimento de *software* e é composta por 3 (três) níveis de abstração, sendo eles:

1. **Nível Conceptual (Objetivo)** – identificar objetivos de medição, listando os objetivos principais do projeto;
2. **Nível Operacional (Questão)** – propor questões para os objetivos de medição, derivando de cada objetivo as questões que devem ser respondidas para determinar se os objetivos são atingidos;
3. **Nível Quantitativo (Métrica)** – definir métricas que ajudem a responder às questões colocadas, decidindo o que deve ser medido por forma a se ser capaz de responder às questões de forma adequada.

A figura 2.9 ilustra o modelo GQM. Esse modelo começa por identificar um objetivo de mediação específico (**nível conceptual**), tendo em conta a entidade, propósito, atributos de qualidade, pontos de vista e ambiente. Os objetivos de medição podem ser:

- **Processos** (testar, desenhar, entrevistar) – coleções de atividades relacionadas com o *software*;
- **Produtos** (programas, testes) – qualquer artefacto, entregáveis ou documentos que resultem de uma atividade de processo;
- **Recursos** (*hardware*, pessoal) – entidades requeridas por uma atividade ou processo.

Cada objetivo é, então, refinado em várias questões (**nível operacional**), que caracterizam a forma como um determinado objetivo poderá ser obtido. Por fim, são identificadas as métricas (**nível quantitativo**), que fornecem a informação quantitativa necessária para

responder às questões do nível anterior. Esta informação pode ser objetiva, se depender apenas do objeto que está a ser medido; ou subjetiva, se depender do objeto e também do ponto de vista do *stakeholder*.

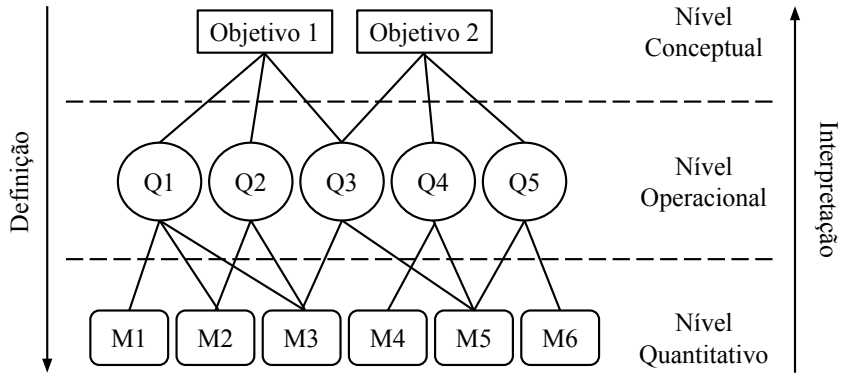


Figura 2.9: Estrutura do modelo QGM

A possibilidade de definir questões abstratas e responder com o auxílio de métricas, a fim de avaliar a qualidade, é relevante para a presente dissertação, uma vez que esta pretende controlar alguns atributos de qualidade, como a complexidade, a completude e a correção.

2.5 Especificação Formal de Métricas

No contexto desta dissertação, torna-se importante especificar as métricas propostas para a avaliação da qualidade dos modelos *i**. Essa especificação, como referido anteriormente, é feita tanto em linguagem natural como em linguagem formal.

Uma linguagem natural, ou seja, uma linguagem desenvolvida naturalmente pelo ser humano, é mais simples e fácil de compreender do que uma linguagem formal. No entanto, é mais propensa a ambiguidades e omissões, fomentando implementações inconsistentes. Por sua vez, uma especificação formal é uma descrição matemática de *software* ou de *hardware* que pode ser utilizada para desenvolver uma implementação dos mesmos. Assim, uma linguagem formal permite evitar eventuais ambiguidades que poderiam resultar em implementações inconsistentes das métricas propostas [Shu+02], e possibilita a automatização do processo de identificação de métricas. A linguagem formal a ser utilizada para a especificação formal de métricas é a OCL.

A OCL (do inglês *Object Constraint Language*) [Obj14a], ou Linguagem para Especificação de Restrições em Objetos, é uma linguagem declarativa utilizada para descrever expressões em modelos UML (do inglês *Unified Modelling Language* [Obj14b]). Essas expressões normalmente especificam condições invariantes que devem ser garantidas no sistema a ser modelado, ou consultas sobre objetos descritos no modelo. Quando as expressões em OCL são avaliadas, não têm qualquer efeito, ou seja, a sua avaliação não pode alterar o estado do sistema onde estão a ser executadas. No entanto, expressões

em OCL podem ser utilizadas para especificar operações/ações que, quando executadas, alteram o estado do sistema [Obj14a].

Esta linguagem é uma linguagem de texto, precisa, que oferece expressões livres das ambiguidades das linguagens naturais, e que possibilita a expressão de restrições em um modelo orientado a objetos que não possam ser especificadas através dos diagramas. Pode ser utilizada com diferentes propósitos, tais como para especificar invariantes em classes e tipos no modelo da classe, para descrever pré e pós condições em operações e métodos, para especificar restrições em operações, entre outros. Apresenta as seguintes vantagens:

- **Melhor documentação** – restrições adicionam informações sobre os elementos dos modelos e as suas relações com os modelos UML;
- **Maior precisão** – restrições em OCL têm uma semântica formal, utilizada para reduzir ambiguidades nos modelos UML;
- **Comunicação sem mal-entendidos** – utilizando restrições em OCL, os modeladores (*modelers*) podem comunicar sem ambiguidades.

A OCL tem 4 (quatro) tipos primitivos de dados: booleanos, inteiros, reais e *strings*. Adicionalmente, tem operadores lógicos (>, <, =, >= e <=) e as declarações são construídas em 4 (quatro) partes, sendo elas:

- **Contexto** – define a situação limita em que a afirmação é válida;
- **Propriedade** – representa algumas características do contexto (por exemplo, se o contexto é uma classe, a propriedade pode ser um atributo);
- **Operação** – podendo ser aritmética ou orientada a conjuntos, manipula ou qualifica uma propriedade;
- **Palavras-chave** – são utilizadas para especificar expressões condicionais (*if*, *then*, *else*, *and*, *or*, *not*, *implies*).

Para melhor se compreender estes conceitos, a figura 2.10 ilustra um diagrama de classes, com as classes `Person`, `Company`, `Job` e `Marriage`.

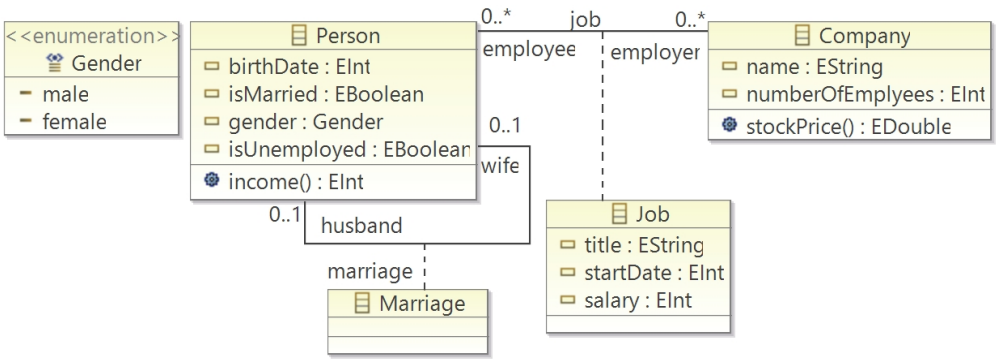


Figura 2.10: Exemplo de diagrama de classes, adaptado de [WK99]

A este diagrama pretende-se acrescentar as seguintes restrições e operações: (i) o número de empregados na empresa tem que ser maior do que 50, (ii) o número de empregrados num determinados trabalho tem que ser maior do que 1 e a sua idade tem que ser superior a 18 anos, (iii) pretende-se saber quem é a mulher de determinada pessoa, e (iv) se uma pessoa estiver desempregada, o seu salário é inferior a 100, caso contrário é igual ou superior a 100.

A listagem 2.1 apresenta o código na linguagem OCL para a especificação das restrições e operações anteriores, pela ordem respetiva.

Listagem 2.1: Exemplo de linguagem OCL, adaptado de [WK99]

```
context Company
  inv: self.numberOfEmployees > 50

context Job
  inv: self.employer.numberOfEmployees >= 1
  inv: self.employee.age >= 18

context Person::getCurrentWife() : Person
  pre: self.isMarried = true and self.gender = male
  post: result = self.marriage.wife

context Person inv:
  let income : Integer = self.job.salary -> sum() in
    if isUnemployed then
      income < 100
    else
      income >= 100
    endif
```

Para além das vantagens apresentadas anteriormente, a OCL foi escolhida como a linguagem formal a ser utilizada nesta dissertação por diversas razões. A primeira prende-se com o facto de que é uma linguagem que oferece formalidade sem sacrificar a compreensão, uma vez que foi desenvolvida tendo em vista a usabilidade para os profissionais de UML. Para além da sua boa ligação com a UML, o facto de as definições serem executáveis, faz com que se evitem problemas de fidelidade na implementação das métricas, que ocorrem frequentemente com outras abordagens [Gou08]. Adicionalmente, é uma abordagem flexível, sendo que para adicionar uma nova métrica, é apenas necessária a definição de uma nova regra OCL que especifica como a métrica deve ser calculada. Por fim, conduz a uma integração direta com o meta-modelo, facilitando o processo de definição de métricas [Abr01; BBA02], e permitindo recolher informação sobre o meta-modelo enquanto se navega sobre o mesmo.

2.6 Sumário

Neste capítulo foram introduzidos conceitos relativos a engenharia de *software*, engenharia de requisitos e engenharia de requisitos orientada a objetivos. Foram apresentadas as três mais importantes metodologias de engenharia de requisitos orientada a objetivos, sendo elas: *framework i**, *framework NFR* e KAOS. A metodologia que foi mais aprofundada, por estar diretamente relacionada com a presente dissertação, foi o *framework i**.

Relativamente a métodos de controlo de qualidade de *software*, foi abordada a técnica GQM, que permite definir métricas para se atingir determinados objetivos de avaliação.

Por fim, em relação a especificação formal de métricas, foi introduzida a linguagem OCL, que permite especificar formalmente as métricas definidas, evitando a existência de eventuais ambiguidades.

3

Trabalho Relacionado

Este capítulo apresenta diversas abordagens relacionadas com o trabalho proposto nesta dissertação: **análise de satisfação de objetivos para modelos de requisitos**, **GO-DKL**, **AIRDoc**, **métricas de integração de *i** com MDD**, **GO-MDD**, ***i*MDF$_M$**, **qualidade nos modelos KAOS** e **avaliação da completude e da granularidade na especificação de requisitos funcionais**. As abordagens avaliam ou preocupam-se com a qualidade de modelos de requisitos, assim como com a definição e especificação de métricas. Para cada uma delas é apresentado um resumo tendo por base a sua descrição original, onde é dado o foco nas principais diferenças e/ou contribuições para esta dissertação. Por fim, é feita uma análise das abordagens.

3.1 Análise de Satisfação de Objetivos

Em [HY12b] é feita uma comparação e avaliação de 7 (sete) abordagens de análise de satisfação de objetivos para modelos de requisitos, no contexto GORE, utilizando algumas ferramentas disponíveis que implementem essas abordagens. As abordagens escolhidas estão relacionadas com GRL (do inglês *Goal-oriented Requirements Language*) [LY04] quantitativo, qualitativo e hídrido; *framework i**; *framework* NFR; e Tropos [GMS05] quantitativo e qualitativo. Os autores avaliam 3 (três) modelos de objetivos exemplificativos, sendo eles: *Media Shop* [CKM01], *Wireless Service Model* [Amy+10] e *Counseling Service Model* [HY09]. Os modelos foram criados utilizando 3 (três) ferramentas distintas, nomeadamente: jUCMNav [RKA06] para a abordagem GRL, OpenOME [HYY11] para as abordagens *i** e NFR, e a ferramenta GR-Tool [Tro14a] para a abordagem Tropos.

Os resultados ajudam a compreender as formas pelas quais as escolhas procedimentais do projeto afetam os resultados de análise de satisfação de objetivos, e como diferenças nos resultados de análise podem levar a diferentes recomendações sobre alternativas de construção do modelo. Segundo os autores, a comparação demonstrou que diferentes técnicas de análise de satisfação de objetivos para modelos de requisitos podem produzir resultados variáveis, dependendo da estrutura do modelo, e que as diferenças nesses resultados são significativas. Assim, recomendam que sejam utilizadas técnicas de análise para modelos de objetivos apenas como heurísticas para tomadas de decisões a nível de desenho do modelo.

Os autores estudaram a análise de satisfação de objetivos e concluíram que, para além de ajudar nas tomadas de decisão a nível de alternativas de desenho, a análise da satisfação de objetivos permite melhorar a qualidade do modelo, aumentar o conhecimento sobre o domínio e facilitar a comunicação. Assim, a satisfação ou a negação de determinados objetivos está diretamente relacionada com a qualidade, o que se prende com a presente dissertação na medida em que o estudo do atributo de qualidade completude pretende, também, verificar até que ponto os objetivos modelados são satisfeitos pelo modelo. Adicionalmente, um dos modelos avaliados é usado como caso de estudo para a avaliação experimental das métricas propostas nesta dissertação.

3.2 GO-DKL

O *framework* GO-DKL (do inglês *Goal-Oriented Design Knowledge Library*) [HY12a] oferece uma abordagem para extrair, codificar e armazenar excertos relacionais de conhecimento de projeto a partir de publicações académicas. É descrito um método para analisar a base de conhecimento, que se destina a ajudar os responsáveis por sistemas de informação a obter, contextualizar e avaliar conteúdos de bases de conhecimento, em relação aos seus próprios projetos. Segundo os autores, tal sistema pode ajudar na criação de novas ideias através de outras já existentes, e apoiar novas "comunidades de prática". Este *framework* foi projetado para fins de reutilização de conhecimento, e foi avaliado por 6 (seis) profissionais de desenho de sistemas de informação, durante entrevistas semi-estruturadas.

O GO-DKL tem 2 (dois) componentes fundamentais: o modelo conceptual e o procedimento de análise. O modelo conceptual orienta a codificação do conhecimento de desenho, e constrói a base da estrutura do repositório da base de dados. Está dividido em 3 (três) secções principais, sendo elas:

- **Dados da biblioteca** – referem-se aos itens de uma publicação, ou seja, as características codificadas do projeto, objetivos, atores e relações entre eles. Os itens estão centrados no conceito de declaração, que é composto por uma ou mais relações entre os itens da biblioteca. Essas relações são avaliadas e podem ser do tipo *helps*, *hurts*, *makes*, *breaks*, *unknown*, *AND* ou *OR*, sendo que a avaliação tem como base a

sintaxe do *framework i**.

- **Dados do projeto** – referem-se à informação que os utilizadores do GO-DKL inserem, selecionam ou manipulam. A sua estrutura é centrada num projeto, que é inicialmente um conjunto de objetivos de alto-nível inseridos na interface do GO-DKL. Os utilizadores podem depois associar os seus objetivos com as categorias da biblioteca e retirar os objetivos que considerem interessantes para os seus projetos. O modelo do projeto é, então, composto por todas as relações entre objetivos adicionados pelos utilizadores e outros objetivos presentes na biblioteca.
- **Dados intermediários** – são utilizados para facilitar a recuperação de itens da base de conhecimento, sendo que o seu principal componente é a agregação de objetivos em categorias. Assim, os utilizadores associam as categorias aos diferentes objetivos do seu projeto.

O procedimento de análise é um método orientado a objetivos que permite recuperar, contextualizar e analisar os itens da biblioteca, com o intuito de utilizar esse conhecimento como ponto de partida para novas configurações, com informações sobre trabalhos anteriores. Este procedimento tem vários passos, cujas atividades podem ser realizadas de forma iterativa.

A figura 3.1 apresenta os passos do procedimento de análise do GO-DKL, e segue-se uma descrição dos mesmos.

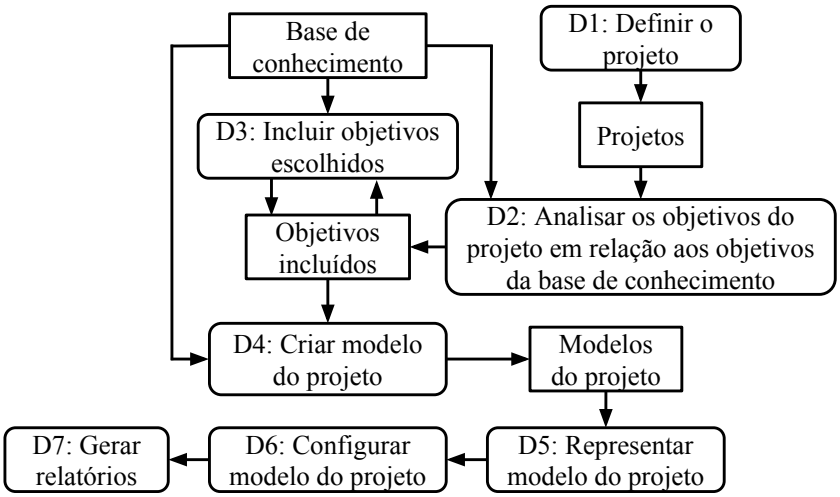


Figura 3.1: Procedimento de análise do GO-DKL, adaptado de [HY12a]

1. **Definir o projeto** – o responsável insere o título do projeto, uma breve descrição do mesmo, e uma lista de objetivos iniciais no sistema de *software*.
2. **Analisar os objetivos do projeto em relação aos objetivos da base de conhecimento** – é escolhido um objetivo do conjunto de objetivos iniciais do projeto, sendo este associado com um ou mais objetivos agregados apresentados pelo sistema de *software*.

3. **Incluir objetivos escolhidos** – são escolhidos objetivos interessantes e relevantes para serem incluídos no modelo do projeto. Os objetivos são divididos em 3 (três) categorias: impactado positivamente, impactado negativamente e prova contraditória.

4. **Criar modelo de projeto** – o modelo é preenchido com todos os itens da biblioteca que contribuem para os objetivos incluídos no passo anterior.

5. **Representar modelo do projeto** – o modelo pode ser representado através de uma estrutura em lista hierárquica em árvore, que oferece uma forma interativa de navegar numa estrutura de dados complexa. Adicionalmente, subconjuntos do modelo que se centram num único objetivo podem ser exportados como ficheiros Q7 [Lei+05] e analisados com o auxílio de uma ferramenta.

6. **Configurar modelo do projeto** – o modelo pode ser configurado para atender a necessidades específicas, sendo que os valores de contribuição entre itens podem ser alterados ou omitidos.

7. **Gerar relatórios** – depois de o modelo ter sido explorado e reconfigurado, podem ser gerados relatórios simples, que servem como referência rápida para o projeto.

Os autores afirmam que os profissionais que avaliaram este *framework* encontraram itens na base de conhecimento que eram extremamente relevantes para os seus projetos. Alguns desses itens tinham sido previamente pensados pelos avaliadores, enquanto que outros surgiram através da análise da base de conhecimento. Devido a este facto, os avaliadores concluíram que este *framework* pode servir como um dispositivo heurístico eficiente, na medida em que funciona como uma lista de controlo que permite analisar questões que ainda não tinham sido consideradas, fazendo com que alguns objetivos que não tenham sido pensados, mas que façam sentido para o projeto em questão, sejam tidos em conta.

Segundo os autores, os avaliadores sentiram que o *framework* podia ser melhorado se os investigadores pudessem contribuir para a base de conhecimento, adicionando novos objetivos, funcionalidades e relacionamentos, fazendo com o sistema se torne uma comunidade *online*. Assim, a presente dissertação pode estender este *framework* através do fornecimento de informações sobre a complexidade, completude e correção dos modelos, permitindo que a base de conhecimento se torne mais completa e, consequentemente, tenha ainda maior utilidade.

3.3 AIRDoc

A abordagem AIRDoc [Ram+08; Ram+11] (do inglês *Approach to Improve Requirements DOCuments*) tem como objetivo facilitar a identificação de possíveis problemas sintáticos

em documentos de requisitos, em particular diagramas de casos de uso, utilizando refabricação e padrões de requisitos. Segundo os autores, a identificação de problemas sintáticos (tal como descrições grandes e pouco claras e informação duplicada) numa fase inicial, assim como a remoção das suas causas, pode melhorar a qualidade de modelos de casos de uso.

Por forma a avaliar modelos de requisitos, e modelos de casos de uso em particular, a abordagem AIRDoc utiliza o modelo GQM e segue 6 (seis) passos, sendo que os 4 (quatro) primeiros correspondem à fase de avaliação, e os 2 (dois) últimos correspondem à fase de melhoria. A figura 3.2 apresenta um diagrama de atividades com esses passos, e segue-se uma descrição dos mesmos.

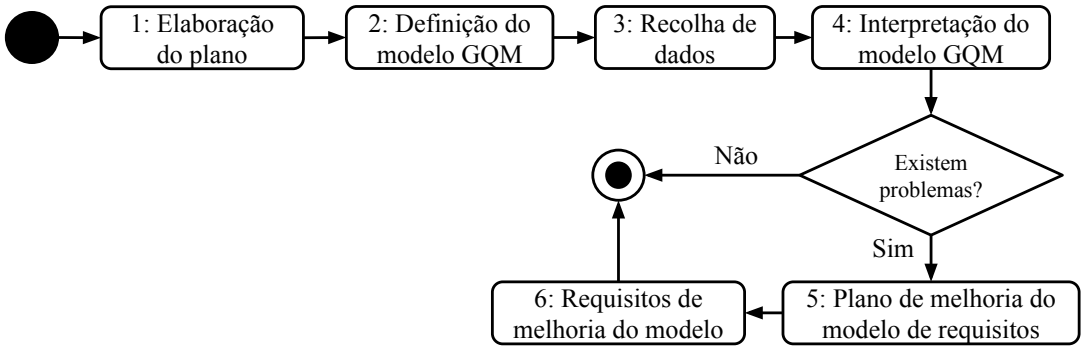


Figura 3.2: Diagrama de atividades do AIRDoc, adaptado de [Ram+08]

1. **Elaboração do plano** – avaliação do modelo de casos de uso, através i) da definição da equipa de qualidade, ii) da escolha de ferramentas e/ou outros recursos, iii) da definição de requisitos de qualidade, e iv) da apresentação do plano de trabalho;

   1.1 **Definição da equipa de qualidade** – seleção dos membros da equipa, sendo que a cada membro deve ser associado pelo menos um papel (engenheiro de requisitos, revisor, entre outros);

   1.2 **Escolha de ferramentas e/ou outros recursos** – definição das ferramentas e/ou recursos a serem utilizados na avaliação, com o respetivo propósito de utilização e tempo de aprendizagem requerido;

   1.3 **Definição dos requisitos de qualidade** – estabelecer quais os requisitos de qualidade a serem avaliados, indicando, para cada um deles, a sua definição, o modelo de requisitos associado, e os requisitos em análise;

   1.4 **Apresentação do plano de trabalho** – documento com as tarefas anteriores explicitadas de forma clara.

2. **Definição do modelo GQM** – definição de todas as medições que vão ser necessárias por forma a garantir os objetivos definidos no passo anterior, através da definição dos requisitos de medição, da elaboração de questões, da definição de métricas,

e da elaboração de hipóteses que irão auxiliar a equipa de qualidade a interpretar o valor das métricas;

3. **Recolha de dados** – para além da recolha de dados, é necessário garantir que as medições definidas pelas métricas especificadas se encontram corretas. Para tal, realiza-se um **período de ensaios**, onde as métricas definidas no passo 2 são testadas; e um **processo de medições e recolha de métricas**, onde os resultados obtidos são analisados e comparados com os limites impostos pela equipa de qualidade;

4. **Interpretação do modelo GQM** – os resultados obtidos nos passos anteriores são analisados e interpretados numa **sessão de *feedback***. Depois, é feito um relatório com os resultados das medições, onde são descritas as observações relevantes, as conclusões e os pontos de ação formulados durante a sessão de *feedback*. Os valores da medição são utilizados para aceitar ou rejeitar as hipóteses;

5. **Plano de melhoria do modelo de requisitos** – os problemas ou sintomas detetados são identificados através do catálogo proposto pela própria abordagem AIRDoc, que fornece uma descrição do problema e das suas possíveis soluções. Após a análise de cada problema, define-se uma solução através da escolha do padrão ou re-fabricação que mais se adequa;

6. **Requisitos de melhoria do modelo** – aplicação dos processos descritos para cada situação ao modelo de requisitos, através da aplicação das soluções selecionadas anteriormente, do armazenamento dos resultados obtidos pelas soluções aplicadas e, por fim, da comparação dos modelos.

Os atributos de qualidade nos quais os autores se centraram são a reutilização e a facilidade de manutenção, sendo diferentes dos propostos nesta dissertação. As métricas não foram definidas formalmente nem implementadas através de uma ferramenta. No entanto, a metodologia seguida é bastante semelhante à da presente dissertação, em particular na utilização da abordagem GQM e por ter como foco a identificação de problemas e a melhoria de modelos de requisitos.

3.3.1 AIRDoc-i*

Em [Sou+12; Sou+13] é proposta a abordagem AIRDoc-i*, uma adaptação do AIRDoc original, que procura identificar erros semânticos e sintáticos em documentos de requisitos *i**, por forma a ajudar o engenheiro de requisitos na utilização dessa linguagem. As principais diferenças relativamente ao AIRDoc prendem-se com o facto de esta abordagem não ser aplicada a modelos de casos de uso mas sim a modelos *i**, e do catálogo de erros frequentes ter sido adaptado em conformidade. Adicionalmente, para além de avaliar modelos *i**, os autores afirmam que esta abordagem também identifica, armazena, quantifica e permite corrigir os erros sintáticos encontrados, através de uma inspeção ao catálogo de erros previamente criado e de uma comparação da sintaxe do modelo *i** com

a do catálogo. No entanto, a correção de erros não é feita de forma automática, ou seja, o engenheiro de requisitos necessita de consultar o catálogo para verificar a forma correta de corrigir o erro e posteriormente aplicar, manualmente, a correção proposta.

O atributo de qualidade no qual os autores se centraram foi a correção, um dos atributos avaliados nesta dissertação. No entanto, esta abordagem também poderá permitir a avaliação de outros atributos de qualidade. Tal como no caso do AIRDoc original, as métricas não foram definidas formalmente nem implementadas através de uma ferramenta, mas foi utilizada a abordagem GQM.

3.4 Métricas de Integração de *i** com MDD

Em [Gia+10] é apresentado um processo de integração de modelos MDD (do inglês *Model-Driven Development*) [Sch06] com modelos orientados a objetivos. Segundo os autores, existem técnicas de modelação que oferecem apoio para elicitação de requisitos e análise de cenários complexos, como é o caso do *framework i**. No entanto, a aplicação desses modelos de requisitos em processos MDD está dependente da experiência dos analistas, para transformar manualmente os modelos de requisitos em modelos MDD apropriados. Têm sido propostas algumas diretrizes por forma a facilitar e automatizar parcialmente esta tradução, mas existe falta de um conjunto de regras de validação sobre como construir modelos *i** para uma geração melhorada dos modelos MDD correspondentes. Assim sendo, os autores propõem um conjunto de métricas que são orientadas para validar a adequação de modelos *i** como ponto de partida para processos MDD baseados em modelos de classes UML, assim como um processo para a aplicação das métricas propostas no *framework i**.

A figura 3.3 apresenta os passos que fazem parte do processo de integração de métricas *i**, e segue-se uma descrição dos mesmos.

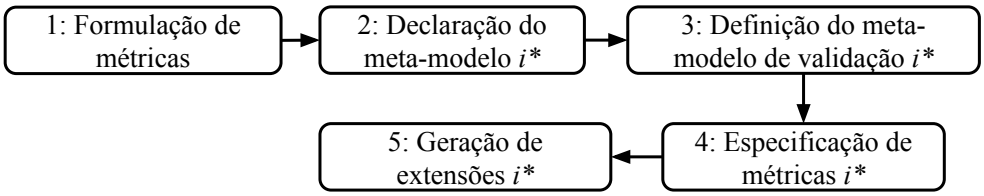


Figura 3.3: Processo de integração de métricas *i**, adaptado de [Gia+10]

1. **Formulação de métricas** – está dividido em 2 (duas) etapas, sendo que a primeira consiste na identificação de elementos do *framework i** que participam no modelo MDD, e a segunda prende-se com a definição das métricas;

2. **Declaração do meta-modelo *i**** – declaração do meta-modelo alvo, sendo que os autores basearam o seu em outras propostas de meta-modelos *i** [Aya+05; Luc+08];

3. **Definição do modelo de validação *i**** – deve incluir a informação necessária que

não conste no meta-modelo definido no ponto anterior;

4. **Especificação de métricas *i**** – corresponde à especificação das métricas através da linguagem OCL, e a integração das mesmas com o modelo de validação *i**. Para a especificação das métricas são aplicados os padrões definidos em [Fra08], que seguem a abordagem GQM. Com o objetivo de completar a definição das métricas, é acrescentada uma nova propriedade que está relacionada com o nível de alerta. Esse nível pode ser crítico, alerta ou informação;

   4.1 **Nível crítico** – indica que a situação identificada pela métrica impede a transformação dos elementos envolvidos na mesma;

   4.2 **Nível alerta** – identifica um problema de modelação que pode ser corrigido por forma a melhorar a geração do modelo;

   4.3 **Nível informação** – indica que é possível acrescentar informações adicionais ao modelo, de maneira a melhorar o processo de geração do mesmo.

5. **Geração de extensões *i**** – utiliza o modelo de validação e a especificação das métricas em OCL para gerar extensões, por forma a integrar as métricas propostas com o *framework i**.

Em comparação com a presente dissertação, este processo centra-se num conjunto diferente de métricas, uma vez que o seu objetivo é auxiliar a avaliação de modelos *i** com uma aproximação MDD baseada em modelos de classes UML. Adicionalmente, a aplicação das métricas não é suportada por uma ferramenta. No entanto, a definição de um meta-modelo *i** e o facto de as métricas serem definidas formalmente, através de OCL, torna-se relevante no contexto desta dissertação.

3.5 GO-MDD

A abordagem GO-MDD [Vas+11; Vas+12] descreve um processo em 6 (seis) fases que integra o *framework i** num processo MDD concreto, aplicando a perspetiva CMMi (do inglês *Capability Maturity Model Integration*) [CMM14]. Os autores afirmam que a engenharia de requisitos orientada a objetivos e o MDD podem ser integrados para atender a exigências de um modelo de maturidade do processo de *software*, a fim de suportar a aplicação de metodologias GORE em cenários da indústria.

A figura 3.4 apresenta as fases que fazem parte da abordagem GO-MDD, e segue-se uma descrição das mesmas.

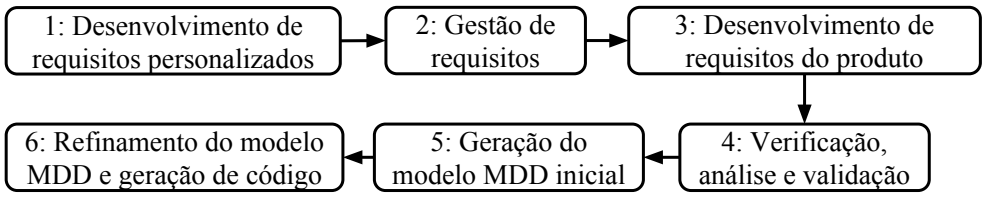


Figura 3.4: Fases da abordagem GO-MDD , adaptado de [Vas+11]

1. **Desenvolvimento de requisitos personalizados** – os requisitos de vários *stakeholders* são consolidados, priorizados de acordo com as necessidades e restrições, e detalhados para serem implementados na iteração atual;

2. **Gestão de requisitos** – os requisitos dos diferentes *stakeholders* são monitorizados, e é realizada a elicitação inicial de requisitos e a sua análise. É tomada a decisão se os requisitos são ou não aprovados para serem desenvolvidos. Esta fase é executada em paralelo com a anterior;

3. **Desenvolvimento de requisitos do produto** – é produzido um modelo SR inicial, através do refinamento do modelo SD do *framework i**. Os objetivos definidos no modelo SR são analisados por forma a se decidir quais os elementos intencionais que devem ser considerados como requisitos do sistema de *software* a ser construído;

4. **Verificação, análise e validação** – conjunto de atividades aplicadas aos modelos definidos nas fases anteriores. Utiliza um conjunto de medidas, especificadas através de regras OCL, que avaliam a completude da geração do modelo MDD em relação aos requisitos especificados no modelo *i**. O conjunto de métricas utilizado é o apresentado em [Ram+08];

5. **Geração do modelo MDD inicial** – depois de o modelo SR do *framework i** ter sido verificado e melhorado, é transformado num modelo de classes inicial através de um conjunto de transformações *model-to-model*;

6. **Refinamento do modelo MDD e geração de código** – o modelo MDD inicial é refinado para introduzir aspetos de desenho que não podem ser obtidos através da transformação anterior. O código do sistema de *software* é gerado, compilado e executado, por forma a ser validado em relação aos requisitos correspondentes.

Tal como na abordagem anterior (presente na secção 3.4), este processo centra-se num conjunto diferente de métricas daquele que é proposto na presente dissertação, uma vez que o seu objetivo é avaliar a completude da geração do modelo MDD em relação aos requisitos especificados no modelo *i**. Adicionalmente, a aplicação das métricas não é suportada por uma ferramenta. No entanto, o facto de as métricas serem definidas formalmente, através de OCL, torna-se relevante no contexto desta dissertação.

3.6 *i*MDF$_M$

O *framework* *i*MDF (do inglês *i* Metrics Definition Framework*) [FG08] procura definir métricas em modelos *i**; analisar a qualidade de modelos individuais; e comparar modelos alternativos em relação a determinadas propriedades, de forma a escolher o modelo alternativo mais apropriado. De notar que as métricas definidas são sobre o que está a ser modelado, ou seja, sobre o domínio, e não sobre o modelo em si. Este *framework* utiliza OCL para formular as métricas, e é composto por:

- **Meta-modelo para o *framework i**** – definição de um meta-modelo onde são representados os conceitos que ajudam no processo de definição de métricas;
- ***Template* de padrões** – definição de padrões, baseados em situações semelhantes, que ajudam na especificação de métricas *i**;
- **Catálogo de padrões** – catalogação dos padrões em relação às características das métricas (estas podem ser métricas de declaração, de definição, de transformação, de modelos auxiliares, entre outros), por forma a facilitar a utilização dos padrões aquando da definição das métricas.

Com o objetivo de melhor conduzir o analista na definição de métricas para modelos *i**, foi proposto o método genérico *i*MDF$_M$ (do inglês *i* Metrics Definition Framework Method*) [Fra08; Fra09], que tem por base o *i*MDF. O método é aplicado para avaliar o desempenho do processo de negócio. A figura 3.5 apresenta os passos que fazem parte deste método, e segue-se uma descrição dos mesmos.

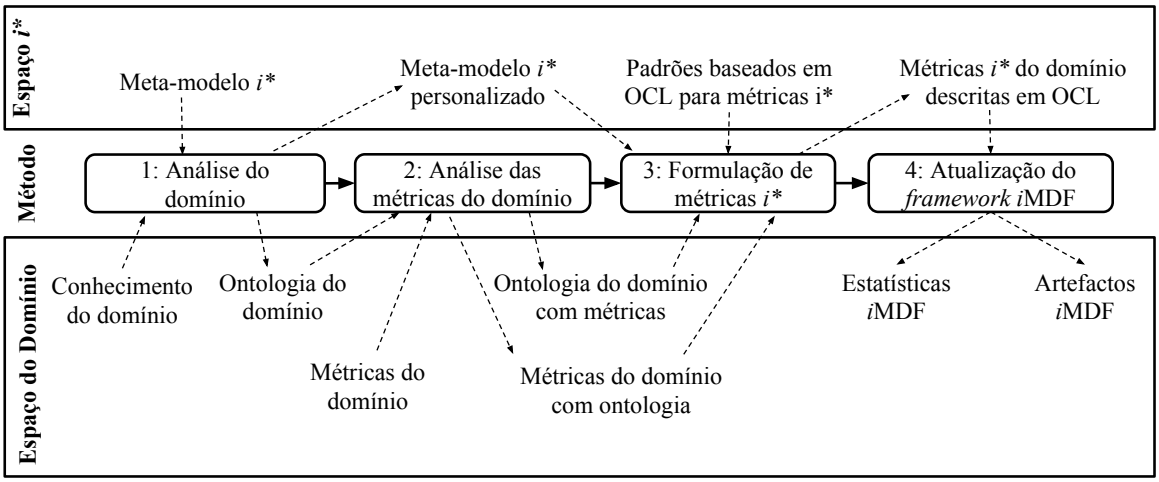


Figura 3.5: Passos do método *i*MDF$_M$, adaptado de [Fra08]

1. **Análise do domínio** – permite compreender o domínio, através da criação de uma ontologia do domínio, do mapeamento da ontologia do domínio para o meta-modelo *i**, e da personalização do meta-modelo *i**, onde podem ser adicionados novos atributos ou classes e impor novas restrições;

2. **Análise das métricas do domínio** – tem como objetivo analisar o conjunto de métricas iniciais do domínio, antes de serem formalizadas. Consiste em estender a ontologia do domínio, por forma a incorporar conceitos que não apareceram na análise anterior; e consolidar o domínio das métricas, através da procura de inconsistências, ambiguidade, falta de uniformidade, entre outros problemas de estruturação;

3. **Formulação de métricas *i**** – torna as métricas operacionais em termos de construções *i**, através do mapeamento das métricas sobre o meta-modelo estendido do *i**, e da expressão das métricas em OCL. De maneira a se expressar as métricas de forma correta, tem-se em conta o diagrama de classes e as restrições de integridade do meta-modelo *i**, utilizando, sempre que possível, os padrões definidos no catálogo da abordagem *i*MDF;

4. **Atualização do *framework* *i*MDF** – processo de atualização das estatísticas relativas, especialmente, à aplicabilidade dos padrões que formam a linguagem; atualização da linguagem dos padrões, uma vez que os padrões que raramente são utilizados podem vir a ser removidos ou reformulados; e atualização do catálogo de métricas. Deve-se decidir se as métricas são ou não adicionadas ao catálogo, ou se é necessário remover alguma métrica desse mesmo catálogo.

Num trabalho mais recente, ao qual foi dado o nome de StarGro [CF14], é proposta a construção de métricas *i** para metodologias AGILE [Agi14], através da utilização do *framework* iMDF. O objetivo desta abordagem é ajudar indivíduos e organizações na melhoria das atividades de gestão de requisitos, através da utilização de métricas para metodologias ágeis em modelos *i**.

Este *framework* está mais centrado no processo e é mais genérico, enquanto que a presente dissertação se foca em aspetos específicos dos modelos *i**. O *i*MDF$_M$ centra-se em métricas dentro do modelo e não sobre o modelo, não sendo o modelo em si a ser avaliado, como nesta dissertação. No entanto, existe o interesse de as métricas serem definidas formalmente, através de OCL.

3.7 Qualidade nos Modelos KAOS

O trabalho presente em [EGA11; Esp12; EGA13] pretende melhorar a qualidade de modelos orientados a objetivos, utilizando como caso de estudo a abordagem KAOS. Para tal, foi definido formalmente um conjunto de métricas, utilizando OCL e seguindo a abordagem GQM, para os atributos de qualidade completude e complexidade.

A avaliação da completude e da complexidade nos modelos KAOS é feita de forma automatizada, com o auxílio de uma ferramenta, o modularKAOS [Dia09; Mon10], tendo a mesma sido estendida por forma a permitir a recolha de informação relevante dos modelos para o cálculo das métricas. Assim, o modularKAOS original foi adaptado para cobrir as necessidades, tendo sido utilizados os *plugins* EMF (do inglês *Eclipse Modeling Framework*) [Ecl14b] e GMF (*Graphical Modeling Framework*) [Ecl14a] do IDE Eclipse [Ecl14j]

para a adaptação e extensão da ferramenta.

Por forma a validar as métricas propostas, foram utilizados 8 (oito) casos de estudo, os quais foram modelados com recurso ao modularKAOS, e aos quais foram aplicadas as métricas definidas. Os resultados foram analisados com recurso a diagramas *boxplot*, que demonstram o grau de dispersão e assimetria dos dados. Segundo os autores, chegou-se à conclusão que, em relação à completude, existem aspetos que não são especificados nesta fase de desenvolvimento, enquanto que a especificação de outros aspetos tem uma grande variação entre os casos de estudo. Relativamente à complexidade, existe alguma consistência, dentro dos casos de estudo, em relação ao que é considerado como um nível de decomposição adequado ao modelo.

Como referido, este trabalho está relacionado com a metodologia KAOS, enquanto que a presente dissertação se centra no *framework i**. No entanto, os objetivos e motivações são semelhantes, assim como a abordagem seguida. Os atributos de qualidade a serem estudados também são semelhantes, sendo que o presente trabalho acrescenta a correção à complexidade e completude.

3.8 Avaliação da Completude e da Granularidade

Em [Esp+09] é apresentada uma experiência controlada sobre a avaliação da qualidade na especificação de requisitos funcionais, tendo como foco a sua completude e a sua granularidade.

Os autores propõem a definição de métricas que permitam medir determinados aspetos relacionados à qualidade de modelos de requisitos, tais como o nível de completude de encapsulamentos funcionais em relação a um modelo de referência, e o número de erros funcionais. Para tal, foi realizada uma experiência com alunos de mestrado, por forma a comparar 2 (duas) abordagens de engenharia de requisitos, sendo elas os casos de uso e a análise de comunicação. Esta última abordagem, não sendo tão conhecida como a primeira, é utilizada com sucesso por diversas empresas espanholas. De forma mais estrita, foram comparados diagramas de casos de uso e diagramas de eventos comunicativos, assim como as suas descrições textuais correspondentes.

A abordagem utilizada no estudo realizado, foi baseada na comparação de um modelo a ser avaliado com um modelo de referência que foi construído e aprovado por uma comissão de especialistas em modelação. As métricas foram definidas seguindo a abordagem GQM, tendo sido descritas textualmente.

Os autores afirmam que o objetivo principal do estudo se prende com a análise da especificação de requisitos funcionais, tendo como propósito a realização de uma avaliação comparativa de técnicas de especificação de requisitos em relação à sua completude e granularidade do ponto de vista do investigador, e no contexto de alunos de mestrados em engenharia informática.

Segundos os autores, os resultados demonstraram que existe uma maior qualidade, em termos de completude e granularidade, quando são seguidas as indicações da análise

de comunicação, do que quando são utilizados os diagramas de casos de uso.

Um dos atributos de qualidade no qual os autores se centraram foi a completude, que também é um dos atributos avaliados nesta dissertação. No entanto, foi realizada uma comparação entre abordagens de engenharia de requisitos, por forma a facilitar a decisão de qual a abordagem a utilizar, e não para melhorar a própria construção do modelo criado com a abordagem escolhida. Apesar de as métricas seguirem a abordagem GQM, as mesmas não foram definidas formalmente nem implementadas através de uma ferramenta.

3.9 Análise das Abordagens

Através deste estudo foi possível analisar abordagens que contribuem para o âmbito desta dissertação, especialmente ao nível da definição e especificação de métricas. A tabela 3.1 apresenta uma comparação entre as diferentes abordagens .

Tabela 3.1: Análise das abordagens estudadas

Abordagem	Modelos suportados	Formulação de métricas	Definição de métricas	Ferramenta
Análise de satisfação de objetivos	GRL, *i**, NFR, Tropos	–	–	Não
GO-DKL	*i**	–	–	Não
AIRDoc	Casos de uso	GQM	Ling. natural	Não
AIRDoc-i*	*i**	GQM	Ling. natural	Não
Integração de *i** com MDD	*i**	GQM	OCL	Não
GO-MDD	*i**	GQM	OCL	Não
*i*MDF$_M$	*i**	*i*MDF	OCL	Não
Qualidade nos modelos KAOS	Kaos	GQM	OCL	Sim
Avaliação da completude e da granularidade	Casos de uso, Análise de comunicação	GQM	Ling. Natural	Não

Uma das principais diferenças entre as abordagens apresentadas, é a forma que cada uma delas tem para criar padrões de métricas. Alguns são especificados através de catálogos, em linguagem natural, enquanto que outros são especificados formalmente, através de OCL. Apesar de a linguagem natural ser mais simples e fácil de compreender, é mais propensa a ambiguidades, omissões e conhecimento tácito, fomentando implementações inconsistentes de instrumentos de medição [Shu+02]. Em contraste, uma linguagem formal permite evitar eventuais ambiguidades e possibilita a automatização do processo de identificação das métricas, sendo bastante útil no contexto desta dissertação.

As abordagens AIRDoc, métricas de integração de *i** com MDD, qualidade nos modelos KAOS, e avaliação da completude e da granularidade na especificação de requisitos funcionais, formalizam as métricas através da abordagem GQM, a técnica utilizada nesta dissertação.

De todas as abordagens relacionadas com métricas, apenas na qualidade nos modelos KAOS as métricas foram implementadas através de uma ferramenta.

Apesar de a análise de satisfação de modelos de requisitos orientados a objetivos, e de o *framework* GO-DKL não estarem relacionadas com a definição e especificação de métricas, são interessantes no contexto desta dissertação, uma vez que se preocupam com a qualidade de modelos de requisitos.

3.10 Sumário

Neste capítulo foram apresentadas diversas abordagens que se preocupam, de formas distintas, com a qualidade de modelos de requisitos, apresentando contributos para a presente dissertação. As diferentes abordagens foram resumidas, sendo apresentados os seus passos principais, quando existentes, e foram descritas as principais diferenças e/ou contribuições para esta dissertação.

4

Métricas para a Avaliação de Modelos *i**

Este capítulo começa por apresentar uma introdução ao conjunto de métricas propostas para a avaliação de modelos *i**, seguindo a abordagem GQM. Em seguida, é apresentado o meta-modelo do *framework i**, necessário para a correta definição formal das métricas. Depois, é apresentada a definição de cada uma das métricas, divididas em métricas para a complexidade, para a completude e para a correção, sendo estruturadas e formalizadas de acordo com o meta-modelo apresentado anteriormente. Por fim, é apresentada a hierarquia completa da definição das métricas, de acordo com a abordagem GQM, por forma a se ter uma visão geral das mesmas.

4.1 Introdução ao Conjunto de Métricas

A definição de métricas, como referido em capítulos anteriores, é realizada seguindo a abordagem GQM. Assim, é necessário ter em conta os níveis conceptual, operacional e quantitativo. O nível conceptual é composto pelos objetivos de avaliar: 1) a complexidade, 2) a completude, e 3) a correção de modelos de requisitos *i**.

As subsecções 4.1.1, 4.1.2 e 4.1.3 apresentam tabelas que sintetizam o resultado da aplicação do nível operacional (primeira coluna) e do nível quantitativo (segunda coluna) da abordagem GQM, por forma a propor um conjunto de métricas que permite satisfazer os objetivos da avaliação da complexidade, completude e correção de modelos *i**, respetivamente. Assim, a primeira coluna apresenta questões que irão permitir avaliar se os objetivos estão a ser alcançados. A segunda coluna apresenta um conjunto de métricas que fornece informação quantitativa para responder à questão correspondente.

4.1.1 Conjunto de Métricas para a Avaliação da Complexidade

Na tabela 4.1, a questão Q1 quantifica a complexidade quando se considera o modelo como um todo, sendo avaliado o seu número de atores e de elementos. As questões Q2–Q5 são direcionadas para avaliar a complexidade dos elementos presentes no modelo, nomeadamente a quantidade de responsabilidades suportadas por um ator (Q2), e o número de decomposições de um objetivo (Q3), *softgoal* (Q4) e tarefa (Q5) de um ator. Para cada uma destas questões, é definida uma métrica base (número de decomposições) e três métricas adicionais, apresentando os valores mínimo, máximo, e médio para a métrica base. As questões Q6 e Q7 quantificam as relações de dependência de um ator, procurando saber se este tem demasiadas dependências de saída (Q6) ou de entrada (Q7), calculando a sua percentagem. Pretende-se saber, também, se determinado ator é um poço ou uma fonte (do inglês *sink* e *source*), ou seja, se tem apenas dependências de entrada ou dependências de saída. Por fim, a questão Q8 permite inferir se a complexidade de um certo ator está relacionada com o seu tipo (ator, agente, posição ou papel), permitindo perceber se existem tipos tendencialmente mais complexos do que outros.

Tabela 4.1: GQM para a avaliação da complexidade

Objetivo: avaliar a complexidade de modelos de requisitos *i**	
Questão	**Métrica**
Q1 – Quão complexo é o modelo, em relação ao número de atores e elementos?	**M1** – Número de atores no modelo **M2** – Número de elementos no modelo
Q2 – Um ator tem demasiada responsabilidade no modelo?	**M3** – Número de elementos na fronteira de atores **M4** – Número mínimo de elementos na fronteira de atores **M5** – Número máximo de elementos na fronteira de atores **M6** – Número médio de elementos na fronteira de atores
Q3 – Quão complexo é um objetivo de um ator, em relação às suas decomposições?	**M7** – Número de decomposições de um objetivo **M8** – Número mínimo de decomposições de um objetivo **M9** – Número máximo de decomposições de um objetivo **M10** – Número médio de decomposições de um objetivo
Q4 – Quão complexo é um *softgoal* de um ator, em relação às suas decomposições?	**M11** – Número de decomposições de um *softgoal* **M12** – Número mínimo de decomposições de um *softgoal* **M13** – Número máximo de decomposições de um *softgoal* **M14** – Número médio de decomposições de um *softgoal*
Q5 – Quão complexa é uma tarefa de um ator, em relação às suas decomposições?	**M15** – Número de decomposições de uma tarefa **M16** – Número mínimo de decomposições de uma tarefa **M17** – Número máximo de decomposições de uma tarefa **M18** – Número médio de decomposições de uma tarefa
Q6 – Um ator é demasiado dependente num modelo?	**M19** – Percentagem de dependências de saída
Q7 – Um ator tem demasiadas dependências num modelo?	**M20** – Percentagem de dependências de entrada
Q8 – Há variação na complexidade média dos diferentes tipos de atores?	**M21** – Número de elementos na fronteira de atores **M22** – Número de elementos na fronteira de agentes **M23** – Número de elementos na fronteira de posições **M24** – Número de elementos na fronteira de papéis

4.1.2 Conjunto de Métricas para a Avaliação da Completude

A completude tem que ver com requisitos que foram identificados nas interações com os *stakeholders*. Na tabela 4.2, a questão Q9 quantifica o detalhe da especificação dos atores, através da percentagem de atores com um tipo específico. As questões Q10 e Q11 quantificam o detalhe a nível dos objetivos e dos *softgoals*, através da percentagem de objetivos com *means-end* (Q10) e da percentagem de *softgoals* com ligações de contribuição (Q11). A questão Q12 quantifica a percentagem de atores com elementos dentro da sua fronteira. Por fim, as questões Q13 e Q14 quantificam quão completo está o modelo, e quão perto se está de se terminar a sua construção, no que diz respeito a atribuição de responsabilidades (Q13), e de dependências ou ligações (Q14) a e entre atores.

Tabela 4.2: GQM para a avaliação da completude

Objetivo: avaliar a completude de modelos de requisitos *i**	
Questão	**Métrica**
Q9 – Quão específicos são os atores?	**M25** – Percentagem de atores com um tipo específico (agente, posição ou papel)
Q10 – Quão detalhados são os objetivos?	**M26** – Percentagem de objetivos com *means-end*
Q11 – Quão detalhados são os *softgoals*?	**M27** – Percentagem de *softgoals* com ligações de contribuição
Q12 – Quão detalhado é o modelo SR em relação aos seus atores?	**M28** – Percentagem de atores com elementos dentro da sua fronteira
Q13 – Quão perto estamos de terminar a atribuição de responsabilidades a um ator?	**M29** – Percentagem de atores sem elementos desconexos dentro da sua fronteira
Q14 – Quão perto estamos de terminar a atribuição de ligações aos atores?	**M30** – Percentagem de atores com ligações de dependência e/ou de associação

4.1.3 Conjunto de Métricas para a Avaliação da Correção

Na tabela 4.3, as questões prendem-se com a correção do modelo. Para se ter uma visão da correção do modelo completo, é necessário analisar cada um dos seus componentes. Assim, as questões Q15-19 avaliam o nível de correção das associações entre atores (Q15), das ligações de dependências (Q16), das ligações de contribuição (Q17), das ligações de decomposição (Q18), e do posicionamento dos atores no modelo (Q19).

Tabela 4.3: GQM para a avaliação da correção

Objetivo: avaliar a correção de modelos de requisitos *i**	
Questão	**Métrica**
Q15 – Quão corretas são as associações?	**M31** – Percentagem de associações corretas
Q16 – Quão corretas são as dependências?	**M32** – Percentagem de dependências corretas
Q17 – Quão corretas são as contribuições?	**M33** – Percentagem de ligações de contribuição corretas
Q18 – Quão corretas são as decomposições?	**M34** – Percentagem de ligações de decomposição corretas
Q19 – Quão correto é o posicionamento dos atores no modelo?	**M35** – Percentagem de atores corretamente posicionados

4.2 Meta-modelo do *Framework i**

Para a criação de um meta-modelo que representasse todas as características do *framework i**, foram consultados alguns já existentes [Aya+05; Luc+08; Ale+08]. Contudo, esses meta-modelos capturam apenas parcialmente a linguagem *i** e, para uma recolha correta das métricas, apresentam problemas em relação a restrições que se pretendem ver satisfeitas. Assim, foi necessário criar um novo, mais adequado às necessidades. Para facilitar a compreensão, o meta-modelo foi dividido em partes, sendo que são apresentadas algumas das mais relevantes. O meta-modelo completo pode ser consultado no anexo A.

Para uma recolha correta das métricas para modelos *i**, é necessário que o meta-modelo possua uma raiz única, a classe `ISTAR`, que vai representar o modelo. Um modelo é composto por nós (`Node`) e relações (`Relationship`). As relações ocorrem entre 2 (dois) nós, sendo que um deles é a origem (`source`) e o outro é o destino (`target`). Por forma a serem identificáveis, tanto o modelo como os diversos nós devem possuir um nome. As diferentes classes e associações entre elas podem ser vistas na figura 4.1.

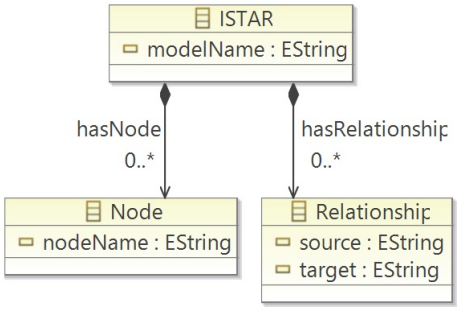


Figura 4.1: Raiz do meta-modelo

Uma relação pode ser uma ligação de dependência (`DependencyLink`), de contribuição (`ContributionLink`), de decomposição (`Decomposition`) ou de associação (`Association`). As decomposições podem ser ligações de decomposição ou ligações de *means-end*. Por uma questão de simplicidade, e exceto no caso das decomposições, apenas são mostradas as classes abstratas e não as subclasses. As diferentes classes e associações entre elas podem ser vistas na figura 4.2.

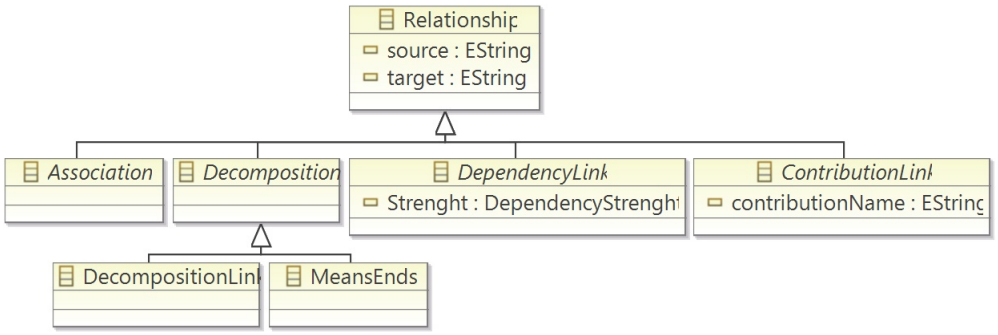


Figura 4.2: Tipos de relações

Um nó pode ser um ator (`Actor`) ou um elemento (`Element`), sendo que um ator pode ter elementos dentro da sua fronteira. No entanto, também existe a possibilidade de os elementos serem criados fora da fronteira de atores. Um ator pode ser especializado em agente (`Agent`), posição (`Position`) ou papel (`Role`). Por forma a serem identificáveis, tanto os atores como elementos devem possuir um nome. As diferentes classes e associações entre elas podem ser vistas na figura 4.3.

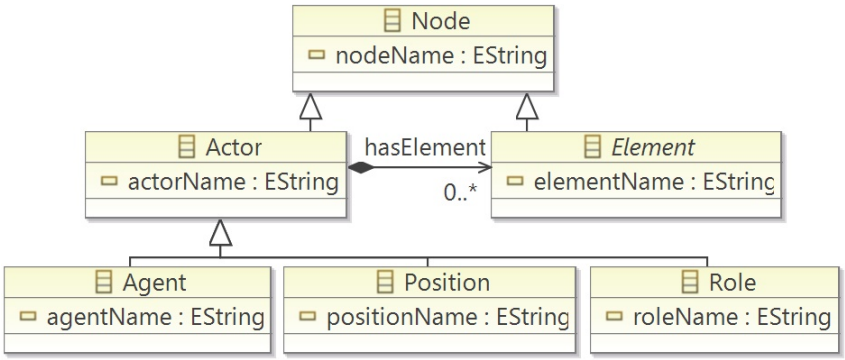


Figura 4.3: Atores e elementos

Por fim, um elemento pode ser de 4 (quatro) tipos: *softgoal* (`Softgoal`), objetivo (`Goal`), tarefa (`Task`), recurso (`Resource`) ou crença (`Belief`), podendo ter ligações de dependência, de contribuição e de decomposição. Excetuando as ligações de dependência, os tipos de ligação não são gerais para todos os elementos, pelo que o tipo de ligação vai depender do tipo de elemento. As diferentes classes e associações entre elas podem ser vistas na figura 4.4.

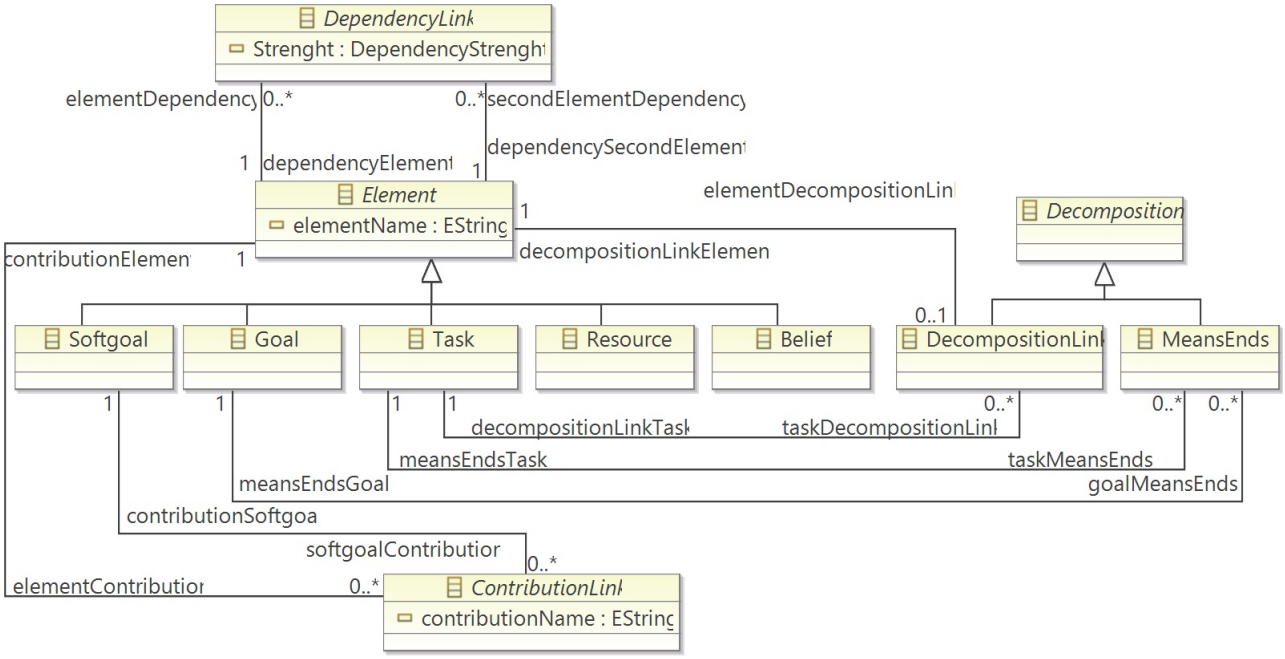


Figura 4.4: Elementos e relações entre elementos

4.3 Definição das Métricas

Com base nas tabelas 4.1, 4.2 e 4.3, que especificam a abordagem GQM para a avaliação de modelos *i** no que respeita à sua complexidade, completude e correção, e tendo em conta o meta-modelo apresentado anteriormente, é possível definir e formalizar o conjunto de métricas.

As métricas estão divididas por questão. Assim, para cada questão são apresentadas as métricas correspondentes. Por sua vez, para cada métrica é apresentado o seu **nome**, uma **definição informal**, em linguagem natural, e uma **definição formal**, em OCL. Sempre que existam, também é dada informação sobre quais as **métricas auxiliares** necessárias para o cálculo correto da métrica em questão. As métricas auxiliares que tenham a sigla MA (Métrica Auxiliar), têm a sua definição apresentada no anexo C.

4.3.1 Métricas para a Complexidade

Da tabela 4.4 à tabela 4.11 são apresentadas as métricas que respondem ao conjunto de questões definido na tabela 4.1, e que diz respeito à avaliação da complexidade dos modelos de requisitos *i**.

Na tabela 4.4 é abordada a questão relativa à complexidade de um modelo SD ou SR, em relação ao seu número de atores e elementos. Por forma a se ter uma primeira noção da complexidade, tendo em conta o modelo como um todo, torna-se importante saber o seu tamanho. Assim, as métricas apresentadas para responder à questão Q1 quantificam o tamanho do modelo, através do seu número de atores (métrica `NAct`) e do seu número de elementos (métrica `NElem`). O tamanho é uma medida útil, que pode ser utilizada para comparar a complexidade entre diferentes modelos. Por exemplo, vários modelos candidatados a um mesmo sistema podem ser comparados, utilizando estas métricas, com respeito à sua complexidade global, por forma a que a tomada de decisão sobre qual o modelo a utilizar seja mais fácil, tendo em conta estes 2 (dois) parâmetros.

Relativamente ao resultado das métricas, é expectável que exista mais do que 1 (um) ator e mais do que 1 (um) elemento no modelo. O número de atores deve variar conforme a complexidade essencial do próprio sistema de *software* que se está a modelar. Em relação ao número de elementos, esse valor pode ser diferente conforme esteja a ser feita uma análise ao modelo SD ou ao modelo SR. No modelo SD é expectável que existam elementos fora da fronteira de atores, mas que não existam elementos dentro da fronteira. No caso do modelo SR, espera-se que exista mais do que 1 (um) elemento fora e mais do que 1 (um) elemento dentro da fronteira de cada um dos atores existentes no modelo.

Tabela 4.4: Definição das métricas correspondentes à questão Q1

Q1 – Quão complexo é o modelo, em relação ao número de atores e elementos?	
Nome	**NAct** – ***N**umber of **Act**ors*
Definição informal	Número total de atores no modelo SD/SR

Definição formal	`context ISTAR`
	`def:NAct():Integer = self.hasNode ->`
	`  select(n:Node | n.oclIsKindOf(Actor)) -> size()`

Nome	**NElem** – ***N**umber of **Elem**ents*
Definição informal	Número total de elementos no modelo SD/SR
Definição formal	`context ISTAR` `def:NElem():Integer = self.NEOAB() + self.NEIAB()`
Necessita	**NEOAB** – ***N**umber of **E**lements **O**utside **A**ctors' **B**oundaries* (MA C.1) **NEIAB** – ***N**umber of **E**lements **I**nside **A**ctors' **B**oundaries*

Na tabela 4.5 é abordada a questão relativa à responsabilidade de um ator num modelo SR. Essa responsabilidade pode ser medida através do número de elementos presentes dentro da fronteira de atores. Tendo em conta o número total de elementos dentro da fronteira de todos os atores (métrica `NEIAB`), torna-se necessário saber qual é o valor mínimo (métrica `MinNEIAB`) de elementos que estão presentes dentro da fronteira de um ator, assim como o valor máximo (métrica `MaxNEIAB`) e o valor médio (métrica `AvgNEIAB`). Também se torna importante saber o número de elementos dentro da fronteira de um ator em particular (métrica `NEI`). De notar que apenas se considera a questão Q2 quando se pretende analisar o modelo SR, uma vez que no modelo SD os atores não possuem elementos dentro da sua fronteira.

O número mínimo, máximo e médio de elementos dentro da fronteira de atores, ajuda o engenheiro de requisitos a reconhecer casos em que a responsabilidade de um ator é superior à esperada. Assim, um valor elevado da métrica `NEI` de um ator, em comparação com a média, pode ser indicador de que esse ator tem muita responsabilidade. De forma análoga, um valor muito baixo em comparação com a média, pode ser indicador de que determinado ator não tem uma importância significativa no modelo. O ideal seria que existisse um equilíbrio entre os atores, e que não houvesse diferenças significativas, tanto para valores demasiado elevados como para valores demasiado baixos. Por fim, a responsabilidade de um ator e, consequentemente, a sua complexidade, também pode ser utilizada para ajudar no cálculo da estimativa de esforço de um projeto de *software*.

Tabela 4.5: Definição das métricas correspondentes à questão Q2

Q2 – Um ator tem demasiada responsabilidade num modelo?	
Nome	**NEIAB** – ***N**umber of **E**lements **I**nside **A**ctors' **B**oundaries*
Definição informal	Número total de elementos dentro da fronteira de todos os atores

Definição formal	`context ISTAR`
	`def:NEIAB():Integer = self.hasNode ->`
	`  select(n:Node | n.oclIsKindOf(Actor)) ->`
	`  iterate(n:Node; total:Integer = 0 |`
	`  let nei:Integer = n.oclAsType(Actor).NEI() in`
	`  total + nei)`

Necessita	**NEI** – ***N**umber of **E**lements **I**nside* (MA C.2)
Nome	**MinNEIAB** – ***Min**imum **N**umber of **E**lements **I**nside **A**ctors' **B**oundaries*
Definição informal	Número mínimo de elementos dentro da fronteira de atores

Definição formal	`context ISTAR`
	`def:MinNEIAB():Integer = self.hasNode ->`
	`  select(n:Node | n.oclIsKindOf(Actor)) ->`
	`  iterate(n:Node; min:Integer = -1 |`
	`  let nei:Integer = n.oclAsType(Actor).NEI() in`
	`  if min = -1 then nei else min.min(nei) endif)`

Necessita	**NEI** – ***N**umber of **E**lements **I**nside* (MA C.2)
Nome	**MaxNEIAB** – ***Max**imum **N**umber of **E**lements **I**nside **A**ctors' **B**oundaries*
Definição informal	Número máximo de elementos dentro da fronteira de atores

Definição formal	`context ISTAR`
	`def:MaxNEIAB():Integer = self.hasNode ->`
	`  select(n:Node | n.oclIsKindOf(Actor)) ->`
	`  iterate(n:Node; max:Integer = -1 |`
	`  let nei:Integer = n.oclAsType(Actor).NEI() in`
	`  if max = -1 then nei else max.max(nei) endif)`

Necessita	**NEI** – ***N**umber of **E**lements **I**nside* (MA C.2)
Nome	**AvgNEIAB** – ***Avg**erage **N**umber of **E**lements **I**nside **A**ctors' **B**oundaries*
Definição informal	Número médio de elementos dentro da fronteira de atores

Definição formal	`context ISTAR::AvgNEIAB()`
	`pre: self.NAct() > 0`
	
	`context ISTAR`
	`def:AvgNEIAB():Double = self.NEIAB() / self.NAct()`

Necessita	**NEIAB** – ***N**umber of **E**lements **I**nside **A**ctors' **B**oundaries* **NAct** – ***N**umber of **A**ctors*

Na tabela 4.6 é abordada a questão relativa à complexidade de um objetivo de um ator no modelo SR, que pode ser medida pelas suas decomposições. Tendo em conta o número total de decomposições de todos os objetivos (métrica `NDAG`), torna-se necessário saber qual é o valor mínimo (métrica `MinNDAG`) de decomposições, o valor máximo (métrica `MaxNDAG`) e o valor médio (métrica `AvgNDAG`). Também se torna importante saber o número de decomposições de um objetivo em particular (métrica `NDGI`). De notar que para responder à questão Q3, o valor mínimo é apenas calculado para objetivos que estejam decompostos. Como tal, objetivos-folha são excluídos do cálculo.

Os valores mínimo, máximo e médio ajudam o engenheiro de requisitos na identificação de complexidades nas decomposições de objetivos que estejam fora do comum. Assim, um valor elevado da métrica `NDGI` de um objetivo, em comparação com a média, pode ser indicador de que esse objetivo possui demasiadas decomposições para uma boa compreensão do mesmo. De forma análoga, um valor muito baixo em comparação com a média, pode ser indicador de que determinado objetivo necessita de uma maior decomposição para ser bem compreendido e, consequentemente, corretamente implementado.

Tabela 4.6: Definição das métricas correspondentes à questão Q3

Q3 – Quão complexo é um objetivo de um ator, em relação às suas decomposições?	
Nome	**NDAG** – ***N**umber of **D**ecompositions **A**ssociated with a **G**oal*

Definição informal	Número total de decomposições associadas a todos os objetivos

Definição formal	`context ISTAR`
	`def:NDAG():Integer = self.hasNode ->`
	`  select(n:Node | n.oclIsKindOf(Actor)) ->`
	`  iterate(n:Node; total:Integer = 0 |`
	`  let ndgi:Integer = n.oclAsType(Actor).NDGI() in`
	`  total + ndgi)`

Necessita	**NDGI** – ***N**umber of **D**ecompositions of a **G**oal **I**nside* (MA C.3)
Nome	**MinNDAG** – ***Min**imum **N**umber of **D**ecompositions **A**ssociated with a **G**oal*
Definição informal	Número mínimo de decomposições associadas a um objetivo

Definição formal	`context ISTAR`
	`def:MinNDAG():Integer = self.hasNode ->`
	`  select(n : Node | n.oclIsKindOf(Actor) and`
	`  n.oclAsType(Actor).MinNDGI() > 0) ->`
	`  iterate(n : Node; min : Integer = -1 |`
	`  let minndgi : Integer = n.oclAsType(Actor).MinNDGI() in`
	`  if min = -1 then minndgi else min.min(minndgi) endif)`

Necessita	**MinNDGI** – ***Min**imum **N**umber of **D**ecompositions of a **G**oal **I**nside* (MA C.5)
Nome	**MaxNDAG** – ***Max**imum **N**umber of **D**ecompositions **A**ssociated with a **G**oal*
Definição informal	Número máximo de decomposições associadas a um objetivo

Definição formal	`context ISTAR`
	`def:MaxNDAG():Integer = self.hasNode ->`
	`  select(n : Node | n.oclIsKindOf(Actor) and`
	`  n.oclAsType(Actor).MaxNDGI() > 0) ->`
	`  iterate(n : Node; max : Integer = -1 |`
	`  let maxndgi : Integer = n.oclAsType(Actor).MaxNDGI() in`
	`  if max = -1 then maxndgi else max.max(maxndgi) endif)`

Necessita	**MaxNDGI** – ***Max**imum **N**umber of **D**ecompositions of a **G**oal **I**nside* (MA C.6)
Nome	**AvgNDAG** – ***A**verage **N**umber of **D**ecompositions **A**ssociated with a **G**oal*
Definição informal	Número médio de decomposições associadas a objetivos

Definição formal	`context ISTAR:AvgNDG()`
	`pre: self.NGWD() > 0`
	
	`context ISTAR`
	`def:AvgNDG():Double = self.NDAG() / self.NGWD()`

Necessita	**NDAG** – ***N**umber of **D**ecompositions **A**ssociated with a **G**oal* **NGWD** – ***N**umber of **G**oals **W**ith **D**ecompositions* (MA C.7)

Na tabela 4.7 é abordada a questão relativa à complexidade de um *softgoal* de um ator no modelo SR, que pode ser medida pelas suas decomposições. Tendo em conta o número total de decomposições de todos os *softgoals* (métrica `NDAS`), torna-se necessário saber qual é o valor mínimo (métrica `MinNDAS`) de decomposições, o valor máximo (métrica `MaxNDAS`) e o valor médio (métrica `AvgNDAS`). Também se torna importante saber o número de decomposições de um *softgoal* em particular (métrica `NDSI`). De notar que para responder à questão Q4, o valor mínimo é apenas calculado para *softgoals* que estejam decompostos. Como tal, *softgoals*-folha são excluídos do cálculo.

Os valores mínimo, máximo e médio ajudam o engenheiro de requisitos na identificação de complexidades nas decomposições de *softgoals* que estejam fora do comum. Assim, um valor elevado da métrica `NDSI` de um *softgoal*, em comparação com a média, pode ser indicador de que esse *softgoal* possui demasiadas decomposições para uma boa compreensão do mesmo. De forma análoga, um valor muito baixo em comparação com a média, pode ser indicador de que determinado *softgoal* necessita de uma maior decomposição para ser bem compreendido e, consequentemente, corretamente implementado.

Tabela 4.7: Definição das métricas correspondentes à questão Q4

Q4 – Quão complexo é um *softgoal* de um ator, em relação às suas decomposições?	
Nome	**NDAS** – ***N**umber of **D**ecompositions **A**ssociated with a **S**oftgoal*
Definição informal	Número total de decomposições associadas a todos os *softgoals*

Definição formal	`context ISTAR`
	`def:NDAS():Integer = self.hasNode ->`
	`  select(n:Node | n.oclIsKindOf(Actor)) ->`
	`  iterate(n:Node; total:Integer = 0 |`
	`  let ndsi:Integer = n.oclAsType(Actor).NDSI() in total+ndsi)`

Necessita	**NDSI** – ***N**umber of **D**ecompositions of a **S**oftgoal **I**nside* (MA C.9)
Nome	**MinNDAS** – ***Min**imum **N**umber of **D**ecompositions **A**ssociated with a **S**oftgoal*
Definição informal	Número mínimo de decomposições associadas a um *softgoal*

Definição formal	`context ISTAR`
	`def:MinNDAS():Integer = self.hasNode ->`
	`  select(n:Node | n.oclIsKindOf(Actor) and`
	`  n.oclAsType(Actor).MinNDSI() > 0) ->`
	`  iterate(n:Node; min:Integer = -1 |`
	`  let minndsi:Integer = n.oclAsType(Actor).MinNDSI() in`
	`  if min = -1 then minndsi else min.min(minndsi) endif)`

Necessita	**MinNDSI** – ***Min**imum **N**umber of **D**ecompositions of a **S**oftgoal **I**nside* (MA C.11)
Nome	**MaxNDAS** – ***Max**imum **N**umber of **D**ecompositions **A**ssociated with a **S**oftgoal*
Definição informal	Número máximo de decomposições associadas a um *softgoal*

Definição formal	`context ISTAR`
	`def:MaxNDAS():Integer = self.hasNode ->`
	`  select(n:Node | n.oclIsKindOf(Actor) and`
	`  n.oclAsType(Actor).MaxNDSI() > 0) ->`
	`  iterate(n:Node; max:Integer = -1 |`
	`  let maxndsi:Integer = n.oclAsType(Actor).MaxNDSI() in`
	`  if max = -1 then maxndsi else max.max(maxndsi) endif)`

Necessita	**MaxNDSI** – ***Max**imum **N**umber of **D**ecompositions of a **S**oftgoal **I**nside* (MA C.12)
Nome	**AvgNDAS** – ***A**verage **N**umber of **D**ecompositions **A**ssociated with a **S**oftgoal*
Definição informal	Número médio de decomposições associadas a um *softgoal*

Definição formal	`context ISTAR::AvgNDAS()`
	`pre: self.NSWD() > 0`
	
	`context ISTAR`
	`def:AvgNDAS():Double = self.NDAS() / self.NSWD()`

Necessita	**NDAS** – ***N**umber of **D**ecompositions **A**ssociated with a **S**oftgoal* **NSWD** – ***N**umber of **S**oftgoals **W**ith **D**ecompositions* (MA C.13)

Na tabela 4.8 é abordada a questão relativa à complexidade de uma tarefa de um ator no modelo SR, que pode ser medida pelas suas decomposições. Tendo em conta o número total de decomposições de todas as tarefas (métrica `NDAT`), torna-se necessário saber qual é o valor mínimo (métrica `MinNDAT`) de decomposições, o valor máximo (métrica `MaxNDAT`) e o valor médio (métrica `AvgNDAT`). Também se torna importante saber o número de decomposições de uma tarefa em particular (métrica `NDTI`). De notar que para responder à questão Q5, o valor mínimo é apenas calculado para tarefas que estejam decompostas. Como tal, tarefas-folha são excluídas do cálculo.

Os valores mínimo, máximo e médio ajudam o engenheiro de requisitos na identificação de complexidades nas decomposições de tarefas que estejam fora do comum. Assim, um valor elevado da métrica `NDTI` de uma tarefa, em comparação com a média, pode ser indicador de que essa tarefa possui demasiadas decomposições para uma boa compreensão da mesma. De forma análoga, um valor muito baixo em comparação com a média, pode ser indicador de que determinada tarefa necessita de uma maior decomposição para ser bem compreendida e, consequentemente, corretamente implementada.

Tabela 4.8: Definição das métricas correspondentes à questão Q5

Q5 – Quão complexa é uma tarefa de um ator, em relação às suas decomposições?	
Nome	**NDAT** – ***N**umber of **D**ecompositions **A**ssociated with a **T**ask*
Definição informal	Número total de decomposições associadas a todas as tarefas

Definição formal	`context ISTAR`
	`def:NDAT():Integer = self.hasNode ->`
	`  select(n:Node | n.oclIsKindOf(Actor)) ->`
	`  iterate(n:Node; total:Integer = 0 |`
	`  let ndti:Integer = n.oclAsType(Actor).NDTI() in total+ndti)`

Necessita	**NDTI** – ***N**umber of **D**ecompositions of a **T**ask **I**nside* (MA C.15)
Nome	**MinNDAT** – ***Min**imum **N**umber of **D**ecompositions **A**ssociated with a **T**ask*
Definição informal	Número mínimo de decomposições associadas a uma tarefa

Definição formal	`context ISTAR`
	`def:MinNDAT():Integer = self.hasNode ->`
	`  select(n:Node | n.oclIsKindOf(Actor) and`
	`  n.oclAsType(Actor).MinNDTI() > 0) ->`
	`  iterate(n:Node; min:Integer = -1 |`
	`  let minndti:Integer = n.oclAsType(Actor).MinNDTI() in`
	`  if min = -1 then minndti else min.min(minndti) endif)`

Necessita	**MinNDTI** – ***Min**imum **N**umber of **D**ecompositions of a **T**ask **I**nside* (MA C.17)
Nome	**MaxNDAT** – ***Max**imum **N**umber of **D**ecompositions **A**ssociated with a **T**ask*
Definição informal	Número máximo de decomposições associadas a uma tarefa

Definição formal	`context ISTAR`
	`def:MaxNDAT():Integer = self.hasNode ->`
	`  select(n:Node | n.oclIsKindOf(Actor) and`
	`  n.oclAsType(Actor).MaxNDTI() > 0) ->`
	`  iterate(n:Node; max:Integer = -1 |`
	`  let maxndti:Integer = n.oclAsType(Actor).MaxNDTI() in`
	`  if max = -1 then maxndti else max.max(maxndti) endif)`

Necessita	**MaxNDTI** – ***Max**imum **N**umber of **D**ecompositions of a **T**ask **I**nside* (MA C.18)
Nome	**AvgNDAT** – ***Av**erage **N**umber of **D**ecompositions **A**ssociated with a **T**ask*
Definição informal	Número médio de decomposições associadas a uma tarefa

Definição formal	`context ISTAR:AvgNDAT()`
	`pre: self.NTWD() > 0`
	
	`context ISTAR`
	`def:AvgNDAT():Integer = self.NDAT() / self.NTWD()`

Necessita	**NDAT** – ***N**umber of **D**ecompositions **A**ssociated with a **T**ask* **NTWD** – ***N**umber of **T**asks **W**ith **D**ecompositions* (MA C.19)

Na tabela 4.9 é abordada a questão relativa às ligações de dependência de um ator no modelo SD ou SR, nomeadamente no que diz respeito às dependências de saída. Assim, a métrica apresentada para responder à questão Q6 (métrica `PODA`) é direcionada para saber se determinado ator está demasiado dependente de outros atores por forma a conseguir atingir os seus objetivos e, em casos extremos, se é uma fonte, ou seja, se apenas tem dependências de saída. A métrica permite identificar situações patológicas e arquétipos de atores no sistema.

A métrica `PODA` tem em conta valores percentuais, sendo que quanto mais alto for o valor da métrica, maior é a probabilidade de o ator estar demasiado dependente de outros. Assim, o valor ideal para a métrica tem que permitir encontrar um equilíbrio entre as dependências de saída e de entrada.

Tabela 4.9: Definição das métricas correspondentes à questão Q6

Q6 – Um ator é demasiado dependente num modelo?	
Nome	**PODA** – ***P**ercentage of **O**utgoing **D**ependencies of **A**ctors*
Definição informal	Percentagem de dependências de saída de todos os atores no modelo SD/SR

Definição formal	`context ISTAR::PODA()`
	`pre: self.NDA() > 0`
	
	`context ISTAR`
	`def:PODA():Double = self.NODA() / self.NDA()`

Necessita	**NODA** – ***N**umber of **O**utgoing **D**ependencies of **A**ctors* (MA C.21) **NDA** – ***N**umber of **D**ependencies of **A**ctors* (MA C.35)

Na tabela 4.10 é abordada a questão relativa às ligações de dependência de um ator no modelo SD ou SR, nomeadamente no que diz respeito às dependências de entrada. Assim, a métrica apresentada para responder à questão Q7 (métrica `PIDA`) quantifica se determinado ator é crucial para que outros atores consigam atingir os seus objetivos e, em casos extremos, se é um poço, ou seja, se apenas tem dependências de entrada.

A métrica `PIDA` tem em conta valores percentuais, sendo que quanto mais alto for o valor da métrica, maior é a probabilidade de o ator ser indispensável para outros. Assim, o valor ideal para a métrica tem que permitir encontrar um equilíbrio entre as dependências de entrada e de saída.

Tabela 4.10: Definição das métricas correspondentes à questão Q7

Q7 – Um ator tem demasiadas dependências num modelo?	
Nome	**PIDA** – ***P**ercentage of **I**ncoming **D**ependencies of **A**ctors*
Definição informal	Percentagem de dependências de entrada de todos os atores no modelo SD/SR

Definição formal	`context ISTAR::PIDA()`
	`pre: self.NDA() > 0`
	
	`context ISTAR`
	`def:PIDA():Double = self.NIDA() / self.NDA()`

Necessita	**NIDA** – ***N**umber of **I**ncoming **D**ependencies of **A**ctors* (MA C.28) **NDA** – ***N**umber of **D**ependencies of **A**ctors* (MA C.35)

Na tabela 4.11 é abordada a questão relativa à variação da complexidade dos diferentes tipos de atores no modelo SR. Assim, as métricas apresentadas para responder à questão Q8 são direcionadas para saber se existe alguma relação entre o tipo de ator (que pode ser *actor*, *agent*, *position* ou *role*) e a sua complexidade a nível de número de elementos dentro da sua fronteira.

As métricas quantificam o número total de elementos dentro da fronteira de todos atores (métrica `NEIActB`), de todos os agentes (métrica `NEIAgentB`), de todas as posições (métrica `NEIPosB`) e de todos os papéis (métrica `NEIRoleB`). Relativamente aos valores, se todas as métricas possuírem valores idênticos pode-se concluir que o tipo de ator e a sua complexidade não estão relacionados. Se, pelo contrário, houver discrepância nos valores das métricas, é possível inferir que a complexidade de determinado ator pode estar relacionada com o seu tipo, e que existem tipos tendencialmente mais complexos do que outros.

Tabela 4.11: Definição das métricas correspondentes à questão Q8

Q8 – Há variação na complexidade média dos diferentes tipos de atores?	
Nome	**NEIActB** – ***N**umber of **E**lements **I**nside **A**ctors' **B**oundaries*
Definição informal	Número total de elementos dentro da fronteira de atores

Definição formal	`context ISTAR`
	`def:NEIActB():Integer = self.NEIAB() - (self.NEIAgentB() +`
	`  self.NEIPosB() + self.NEIRoleB())`

Necessita	**NEIAB** – ***N**umber of **E**lements **I**nside **A**ctors' **B**oundaries* **NEIAgentB** – ***N**umber of **E**lements **I**nside **Agent**s' **B**oundaries* **NEIPosB** – ***N**umber of **E**lements **I**nside **Pos**itions' **B**oundaries* **NEIRoleB** – ***N**umber of **E**lements **I**nside **Role**s' **B**oundaries*
Nome	**NEIAgentB** – ***N**umber of **E**lements **I**nside **Agent**s' **B**oundaries*
Definição informal	Número total de elementos dentro da fronteira de agentes

Definição formal	`context ISTAR`
	`def:NEIAgentB():Integer = self.hasNode ->`
	`  select(n:Node | n.oclIsKindOf(Agent)) ->`
	`  iterate(n:Node; total:Integer = 0 |`
	`  let neia:Integer = n.oclAsType(Agent).NEIA() in total+neia)`

Necessita	**NEIA** – ***N**umber of **E**lements **I**nside an **A**gent* (MA C.39)
Nome	**NEIPosB** – ***N**umber of **E**lements **I**nside **P**ositions' **B**oundaries*
Definição informal	Número total de elementos dentro da fronteira de posições

Definição formal	`context ISTAR`
	`def:NEIPosB():Integer = self.hasNode ->`
	`select(n:Node | n.oclIsKindOf(Position)) ->`
	`  iterate(n:Node; total:Integer = 0 |`
	`  let neip:Integer = n.oclAsType(Position).NEIP() in total+neip)`

Necessita	**NEIP** – ***N**umber of **E**lements **I**nside a **P**osition* (MA C.40)
Nome	**NEIRoleB** – ***N**umber of **E**lements **I**nside **R**oles' **B**oundaries*
Definição informal	Número total de elementos dentro da fronteira de papéis

Definição formal	`context ISTAR`
	`def:NEIRoleB():Integer = self.hasNode ->`
	`  select(n:Node | n.oclIsKindOf(Role)) ->`
	`  iterate(n:Node; total : Integer = 0 |`
	`  let neir : Integer = n.oclAsType(Role).NEIR() in total+neir)`

Necessita	**NEIR** – ***N**umber of **E**lements **I**nside a **R**ole* (MA C.41)

4.3.2 Métricas para a Completude

Da tabela 4.12 à tabela 4.18 são apresentadas as métricas que respondem ao conjunto de questões definido na tabela 4.2, e que diz respeito à avaliação da completude dos modelos de requisitos *i**.

Na tabela 4.12 é abordada a questão relativa à especificação de atores no modelo SD ou SR. Assim, a métrica apresentada para responder à questão Q9 (métrica `PSAct`) quantifica a percentagem de atores com um tipo específico, ou seja, a percentagem de atores especializados. Os atores são um tipo geral, enquanto que agentes, papéis e posições são mais específicos.

Segundo o guia de boas práticas do *framework i**, presente na página *wiki* [wik14c], deve-se utilizar a notação especializada, uma vez que há uma maior precisão e detalhe quando se modela atores como agentes, papéis ou posições, sempre que a distinção entre eles seja clara. Quão mais detalhado for um modelo, mais completo ele se torna.

A utilização de tipos específicos pode facilitar a obtenção de um maior nível de detalhe através da instanciação de *stakeholders* reais e captando o conhecimento de domínio. A falta de utilização de qualquer uma das noções de atores específicos pode fazer com que o modelo perca alguma informação útil. No entanto, e ainda de acordo com a página *wiki*, o uso excessivo desta notação específica pode fazer com que os modelos se tornem muito mais complexos e difíceis de lidar e compreender, o que entra em conflito com o objetivo de reduzir a complexidade acidental. Assim, recomenda-se a utilização de tipos específicos de atores tendo em conta o valor e a informação adicional que estes vão acrescentar ao modelo [wik14b]. Desta forma, o valor ideal para a métrica `PSAct` irá depender do modelo do sistema de *software* em questão. No entanto, é uma medida útil, que pode ser utilizada para comparar a especialização de atores entre diferentes modelos, e avaliar se existe algum efeito de uma maior precisão no nível de especialização dos atores.

Tabela 4.12: Definição das métricas correspondentes à questão Q9

Q9 – Quão específicos são os atores?	
Nome	**PSAct** – ***P**ercentage of **S**pecific **A**ctors*
Definição informal	Percentagem de atores com um tipo específico (agente, papel ou posição)

Definição formal	`context ISTAR::PSAct()`
	`pre: self.NAct() > 0`
	
	`context ISTAR`
	`def:PSAct():Double = (self.NAgents() + self.NRoles() +`
	`  self.NPos()) / self.NAct()`

Necessita	**NAgents** – ***N**umber of **Agents*** (MA C.42) **NRoles** – ***N**umber of **Roles*** (MA C.43) **NPos** – ***N**umber of **Pos**itions* (MA C.44) **NAct** – ***N**umber of **Act**ors*

Na tabela 4.13 é abordada a questão relativa ao nível de detalhe dos objetivos no modelo SR. Assim, a métrica apresentada para responder à questão Q10 (métrica `PGWME`) quantifica a percentagem de objetivos com decomposições, ou seja, com ligações de *means-end* associadas.

No modelo SR, elementos que não estejam decompostos em outros elementos e/ou que não recebam nenhuma ligação de contribuição são considerados elementos-folha. No entanto, os objetivos são considerados elementos de alto nível, pelo que necessitam de ser refinados, e só o podem ser através da utilização de ligações de *means-end*, onde o meio para atingir o objetivo é uma tarefa. A existência de várias ligações de *means-end* associadas a um objetivo indicam alternativas para a satisfação desse mesmo objetivo.

A métrica `PGWME` tem em conta valores percentuais, sendo que quanto mais alto for o valor da métrica, maior é o nível de detalhe dos objetivos presentes no modelo. Sendo que todos os objetivos devem ser decompostos, o valor ideal para a métrica será de 100%.

Tabela 4.13: Definição das métricas correspondentes à questão Q10

Q10 – Quão detalhados são os objetivos?	
Nome	**PGWME** – ***P**ercentage of **G**oals **W**ith **M**eans-end*
Definição informal	Percentagem de objetivos com *means-end* no modelo SD/SR

Definição formal	`context ISTAR::PGWME()`
	`pre: self.NGIAB() > 0`
	
	`context ISTAR`
	`def:PGWME():Double = self.NGWD() / self.NGIAB()`

Necessita	**NGWD** – ***N**umber of **G**oals **W**ith **D**ecompositions* (MA C.7) **NGIAB** – ***N**umber of **G**oals **I**nside **A**ctors' **B**oundaries* (MA C.45)

Na tabela 4.14 é abordada a questão relativa ao nível de detalhe dos *softgoals* no modelo SR. Assim, a métrica apresentada para responder à questão Q11 (métrica `PSWC`)

quantifica a percentagem de *softgoals* com decomposições, ou seja, com ligações de contribuição.

Tal como no caso dos objetivos (apresentado na questão Q10), os *softgoals* são considerados elementos de alto nível, pelo que necessitam de ser refinados e operacionalizados, e só o podem ser através da utilização de ligações de contribuição. A operacionalização é feita através de tarefas de baixo nível, ou seja, tarefas-folha.

A métrica `PSWC` tem em conta valores percentuais, sendo que quanto mais alto for o valor da métrica, maior é o nível de detalhe dos *softgoals* presentes no modelo. Sendo que todos os *softgoals* devem ser decompostos, o valor ideal para a métrica será de 100%.

Tabela 4.14: Definição das métricas correspondentes à questão Q11

Q11 – Quão detalhados são os *softgoals*?	
Nome	**PSWC** – ***P**ercentage of **S**oftgoals **W**ith **C**ontributions*
Definição informal	Percentagem de *softgoals* com ligações de contribuição no modelo SD/SR

Definição formal	`context ISTAR::PSWC()`
	`pre: self.NSIAB() > 0`
	
	`context ISTAR`
	`def:PSWC():Double = self.NSWD() / self.NSIAB()`

Necessita	**NSWD** – ***N**umber of **S**oftgoals **W**ith **D**ecompositions* (MA C.13) **NSIAB** – ***N**umber of **S**oftgoals **I**nside **A**ctors' **B**oundaries* (MA C.47)

Ainda relativamente às questões Q10 e Q11, vale a pena referir que estão a ser considerados objetivos e *softgoals*, respetivamente, mas não existe nenhuma questão em relação às decomposições de uma tarefa ou de uma crença. As crenças, pela sua definição, não têm decomposições, mas as tarefas podem ser decompostas através de ligações de decomposição. No entanto, a nível de completude, o facto de uma tarefa não ser decomposta não oferece nenhuma informação útil. Todas as tarefas de baixo nível (folha) constituem as necessidades reais do sistema. Não obstante, o nível e a profundidade do refinamento depende de alguns fatores, como o esforço que tem que ser despendido da modelação, a escalabilidade do modelo e o objetivo do modelo em si. Assim sendo, o modelador pode querer tentar chegar a requisitos de baixo nível, ou avaliar as opções a mais alto nível, sem que isso interfira na incompletude do seu modelo.

Na tabela 4.15 é abordada a questão relativa ao nível de detalhe de um modelo SR, tendo em conta os elementos presentes na fronteira dos seus atores. Assim, a métrica apresentada para responder à questão Q12 (métrica `PAWEI`) quantifica a percentagem de atores com elementos dentro da sua fronteira. No modelo SR, considera-se que atores que não tenham elementos na sua fronteira não estão a ser suficientemente detalhados, e a sua presença no modelo pode não ser necessária, uma vez que podem não oferecer nenhum tipo de informação relevante.

A métrica `PAWEI` tem em conta valores percentuais, sendo que quanto mais alto for o valor da métrica, maior é o nível de detalhe dos atores presentes no modelo. Sendo que

todos os atores devem ter elementos dentro da sua fronteira no modelo SR, o valor ideal para a métrica será de 100%.

Tabela 4.15: Definição das métricas correspondentes à questão Q12

Q12 – Quão detalhado é o modelo SR em relação aos seus atores?	
Nome	**PAWEI** – *Percentage of Actors With Elements Inside*
Definição informal	Percentagem de atores com elementos dentro da sua fronteira

Definição formal	`context ISTAR::PAWEI()`
	`pre: self.NAct() > 0`
	
	`context ISTAR`
	`def:PAWEI():Double = self.NAWEI() / self.NAct()`

Necessita	**NAWEI** – *Number of Actors With Elements Inside* (MA C.49) **NAct** – *Number of Actors*

Na tabela 4.16 é abordada a questão relativa a quão perto se está de se terminar a construção de determinado modelo SR, nomeadamente no que diz respeito a atribuição de responsabilidades a atores. Os elementos presentes na fronteira de um ator têm que possuir pelo menos uma ligação de algum tipo, seja ele decomposição, *means-end*, contribuição ou dependência, ou seja, não podem existir elementos desconexos. Assim, a métrica apresentada para responder à questão Q13 (métrica `PAWOUEI`) quantifica a percentagem de atores sem elementos desconexos dentro da sua fronteira.

A métrica `PAWOUEI` tem em conta valores percentuais, sendo que quanto mais alto for o valor da métrica, mais perto se está de se terminar a atribuição de responsabilidades a atores presentes no modelo. Sendo que todos os elementos dentro da fronteira de atores têm que ter pelo menos uma ligação, o valor ideal para a métrica será de 100%.

Tabela 4.16: Definição das métricas correspondentes à questão Q13

Q13 – Quão perto estamos de terminar a atribuição de responsabilidades a um ator?	
Nome	**PAWOUEI** – *Percentage of Actors WithOut Unconnected Elements Inside*
Definição informal	Percentagem de atores sem elementos desconexos dentro da sua fronteira
Definição formal	`context ISTAR` `def:PAWOUEI():Double = 1 - self.PAWUEI()`
Necessita	**PAWUEI** – *Percentage of Actors With Unconnected Elements Inside* (MA C.50)

Na tabela 4.17 é abordada a questão relativa a quão perto se está de se terminar a atribuição de ligações entre atores no modelo SD ou SR. Existem duas formas de se ligar atores: através de ligações de dependência e através de ligações de associação. Um dos objetivos do modelo SR é mostrar e analisar como as dependências são satisfeitas entre atores. Quando os elementos internos de um ator são conhecidos, as dependências devem ligar ao elemento interno e não ao ator, ajudando na construção de modelos mais precisos e com maior utilidade. As ligações de associação permitem descrever as relações entre atores. Assim, a métrica apresentada para responder à questão Q14 (métrica

PAWDOA) é direcionada para calcular a percentagem de atores com ligações de dependência e/ou de associação.

A métrica PAWDOA tem em conta valores percentuais, sendo que quanto mais alto for o valor da métrica, mais perto se está de se terminar a atribuição de ligações a atores presentes no modelo. Sendo que todos os atores devem ter pelo menos 1 (uma) ligação, o valor ideal para a métrica será de 100%.

Tabela 4.17: Definição das métricas correspondentes à questão Q14

Q14 – Quão perto estamos de terminar a atribuição de ligações aos atores?	
Nome	**PAWDOA** – ***P**ercentage of **A**ctors **W**ith **D**ependencies **O**r **A**ssociations*
Definição informal	Percentagem de atores com ligações de dependência e/ou de associação

Definição formal	`context ISTAR::PAWDOA()`
	`pre: NAct() > 0`
	
	`context ISTAR`
	`def:PAWDOA():Double = self.NAWDOA() / self.NAct()`

Necessita	**NAWDOA** – ***N**umber of **A**ctors **W**ith **D**ependencies **O**r **A**ssociations* (MA C.73) **NAct** – ***N**umber of **A**ctors*

4.3.3 Métricas para a Correção

Da tabela 4.18 à tabela 4.22 são apresentadas as métricas que respondem ao conjunto de questões definido na tabela 4.3, e que diz respeito à avaliação da correção dos modelos de requisitos *i**.

Na tabela 4.18 é abordada a questão relativa à correção das associações entre atores no modelo SD ou SR. Assim, a métrica apresentada para responder à questão Q15 (métrica PCA) quantifica a percentagem de ligações de associação corretas entre atores.

A métrica PCA tem em conta valores percentuais, sendo que quanto mais alto for o valor da métrica, maior é o nível de correção das ligações de associação, pelo que o valor ideal para a métrica será de 100%.

Como descrito na secção 2.3.1, referente ao *framework i**, as ligações de associação são utilizadas para descrever as relações entre atores e, para que possam ser consideradas corretas, têm que respeitar as seguintes regras:

- As ligações *ISA* e *Is part of* apenas podem ser utilizadas entre atores do mesmo tipo específico, ou seja, ator–ator, agente–agente, posição–posição e papel–papel;
- A ligação *Covers* apenas pode ser utilizada entre posições e papéis, sendo que a origem é a posição e o destino é o papel;
- A ligação *Occupies* apenas pode ser utilizada entre agentes e posições, sendo que a origem é o agente e o destino é a posição;
- A ligação *Plays* apenas pode ser utilizada entre agentes e papéis, sendo que a origem é o agente e o destino é o papel;

- A ligação *INS* apenas pode ser utilizada entre agentes, onde a origem e o destino são 2 (dois) agentes diferentes.

A utilização de ligações de associações de forma diferente da especificada, é considerada uma utilização incorreta.

Tabela 4.18: Definição das métricas correspondentes à questão Q15

Q15 – Quão corretas são as associações?	
Nome	**PCA** – ***P**ercentage of **C**orrect **A**ssociations*
Definição informal	Percentagem de associações corretas

Definição formal	`context ISTAR::PCA()`
	`pre: self.TNA() > 0`
	
	`context ISTAR`
	`def:PCA():Double = self.NCA() / self.TNA()`

Necessita	**NCA** – ***N**umber of **C**orrect **A**ssociations* **TNA** – ***T**otal **N**umber of **A**ssociations* (MA C.77)

Na tabela 4.19 é abordada a questão relativa à correção das dependências entre atores no modelo SD ou SR. Assim, a métrica apresentada para responder à questão Q16 (métrica `PCD`) quantifica a percentagem de ligações de dependência corretas entre atores.

A métrica `PCD` tem em conta valores percentuais, sendo que quanto mais alto for o valor da métrica, maior é o nível de correção das ligações de dependência, pelo que o valor ideal para a métrica será de 100%.

Como descrito na secção 2.3.1, referente ao *framework i**, as ligações de dependência indicam que um ator depende de outro para realizar algo e, para que possam ser consideradas corretas, têm que respeitar as seguintes regras:

- Têm que existir sempre 3 (três) partes numa ligação de dependência: o *depender*, o *dependee* e o *dependum*;
- As ligações devem ter a mesma direção:
  - *Depender* como origem e *dependum* como destino;
  - *Dependum* como origem e *dependee* como destino.
- Não podem ser utilizadas para ligar elementos dentro da fronteira de um ator;
- Um elemento pode ser *depender*, se estiver dentro da fronteira de um ator;
- Um *softgoal* nunca deve depender de um objetivo;
- Os *dependums* não devem ser utilizados em mais do que uma ligação de dependência, uma vez que resulta em ambiguidade.

A utilização de ligações de dependência de forma diferente da especificada, é considerada uma utilização incorreta.

Tabela 4.19: Definição das métricas correspondentes à questão Q16

Q16 – Quão corretas são as dependências?	
Nome	**PCD** – ***P**ercentage of **C**orrect **D**ependencies*
Definição informal	Percentagem de ligações de dependência corretas

Definição formal	`context ISTAR::PCD()`
	`pre: self.TND() > 0`
	
	`context ISTAR`
	`def:PCD():Double = self.NCD() / self.TND()`

Necessita	**NCD** – ***N**umber of **C**orrect **D**ependencies* **TND** – ***T**otal **N**umber of **D**ependencies* (MA C.78)

Na tabela 4.20 é abordada a questão relativa à correção das contribuições no modelo SR. Assim, a métrica apresentada para responder à questão Q17 (métrica `PCC`) quantifica a percentagem de ligações de contribuição corretas.

A métrica `PCC` tem em conta valores percentuais, sendo que quanto mais alto for o valor da métrica, maior é o nível de correção das ligações de contribuição, pelo que o valor ideal para a métrica será de 100%.

As ligações de contribuição permitem indicar como é que um elemento contribui para alcançar os atributos de qualidade especificados pelos *softgoals* e, para que possam ser consideradas corretas, têm que respeitar as seguintes regras:

- A origem da ligação apenas pode ser um elemento;
- O destino da ligação apenas pode ser um *softgoal*;
- Tanto a origem com o destino têm que se encontrar dentro da fronteira de atores.

A utilização de ligações de contribuição de forma diferente da especificada, é considerada uma utilização incorreta.

Tabela 4.20: Definição das métricas correspondentes à questão Q17

Q17 – Quão corretas são as contribuições?	
Nome	**PCC** – ***P**ercentage of **C**orrect **C**ontributions*
Definição informal	Percentagem de ligações de contribuição corretas

Definição formal	`context ISTAR::PCC()`
	`pre: TNCL() > 0`
	
	`context ISTAR`
	`def:PCC():Double = self.NCC() / self.TNC()`

Necessita	**NCC** – ***N**umber of **C**orrect **C**ontributions* **TNC** – ***T**otal **N**umber of **C**ontributions* (MA C.79)

Na tabela 4.21 é abordada a questão relativa à correção das decomposições no modelo SD ou SR. Assim, a métrica apresentada para responder à questão Q18 (métrica `PCDL`) quantifica a percentagem de ligações de decomposição corretas.

A métrica `PCDL` tem em conta valores percentuais, sendo que quanto mais alto for o valor da métrica, maior é o nível de correção das ligações de decomposição, pelo que o valor ideal para a métrica será de 100%.

As ligações de decomposição permitem especificar uma tarefa e, para que possam ser consideradas corretas, têm que respeitar as seguintes regras:

- A origem da ligação apenas pode ser uma tarefa;
- O destino da ligação apenas pode ser uma uma tarefa, um objetivo, um recurso ou um *softgoal*;
- Tanto a origem como o destino têm que se encontrar dentro da fronteira de atores.

A utilização de ligações de decomposição de forma diferente da especificada, é considerada uma utilização incorreta.

Tabela 4.21: Definição das métricas correspondentes à questão Q18

Q18 – Quão corretas são as decomposições?	
Nome	**PCDL** – ***P**ercentage of **C**orrect **D**ecomposition **L**inks*
Definição informal	Percentagem de ligações de decomposição corretas

Definição formal	`context ISTAR::PCDL()`
	`pre: self.TNDL() > 0`
	
	`context ISTAR`
	`def:PCDL():Double = self.NCDL() / self.TNDL()`

Necessita	**NCDL** – ***N**umber of **C**orrect **D**ecomposition **L**inks* **TNDL** – ***T**otal **N**umber of **D**ecomposition **L**inks* (MA C.80)

Na tabela 4.22 é abordada a questão relativa à correção do posicionamento dos atores no modelo SD ou SR. Assim, a métrica apresentada para responder à questão Q19 (métrica `PCAct`) quantifica a percentagem de atores corretamente posicionados.

A métrica `PCAct` tem em conta valores percentuais, sendo que quanto mais alto for o valor da métrica, maior é o nível de correção do posicionamento dos atores, pelo que o valor ideal para a métrica será de 100%.

Para que o posicionamento de um ator no modelo possa ser considerado correto, este não se pode encontrar dentro da fronteira de outro ator. Se tal se verificar, o posicionamento é considerado incorreto.

Tabela 4.22: Definição das métricas correspondentes à questão Q19

Q19 – Quão correto é o posicionamento dos atores no modelo?	
Nome	**PCAct** – ***P**ercentage of **C**orrect **Act**ors*
Definição informal	Percentagem de atores corretamente posicionados

Definição formal	**context** ISTAR::PCAct() **pre**: self.NAct() > 0 **context** ISTAR def:PCAct():Double = self.NCAct() / self.NAct()
Necessita	**NCAct** – *Number of Correct Actors* **NAct** – *Number of Actors*

4.4 Diagrama GQM

As figuras 4.5, 4.6 e 4.7 ilustram o resultado final da aplicação da abordagem GQM. Os diagramas devem ser vistos como um todo, mas foram separados por objetivo, para uma melhor compreensão e leitura. As métricas auxiliares não são apresentadas nos diagramas devido ao seu elevado número.

Para a definição do objetivo complexidade, apresentado na figura 4.5, são necessárias 8 (oito) questões e 24 (vinte e quatro) métricas, numa média de 3 (três) métricas por questão. Este objetivo apresenta o maior número de questões e métricas.

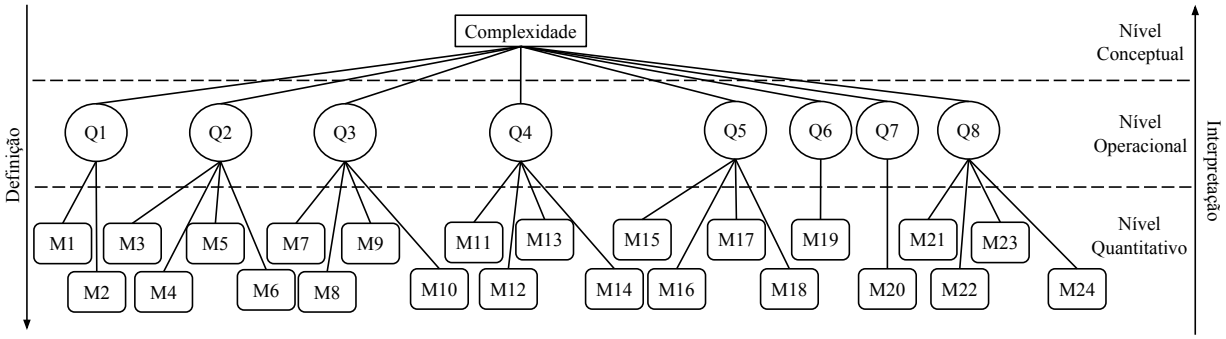


Figura 4.5: Aplicação da Abordagem GQM - Complexidade

Para a definição do objetivo completude, apresentado na figura 4.6, são necessárias 6 (seis) questões e 6 (seis) métricas, numa média de 1 (uma) métrica por questão.

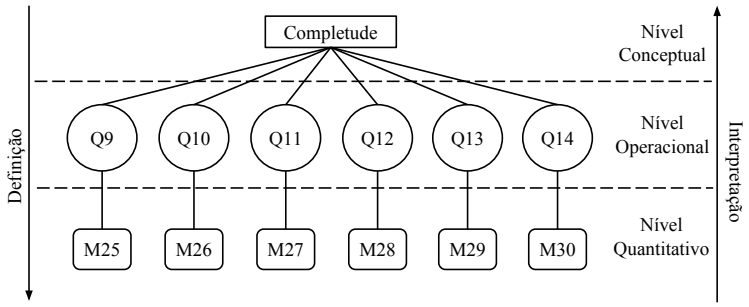


Figura 4.6: Aplicação da Abordagem GQM - Completude

Para a definição do objetivo correção, apresentado na figura 4.7, são necessárias 5 (cinco) questões e 5 (cinco) métricas, numa média de 1 (uma) métrica por questão. Este

objetivo apresenta o menor número de questões e métricas.

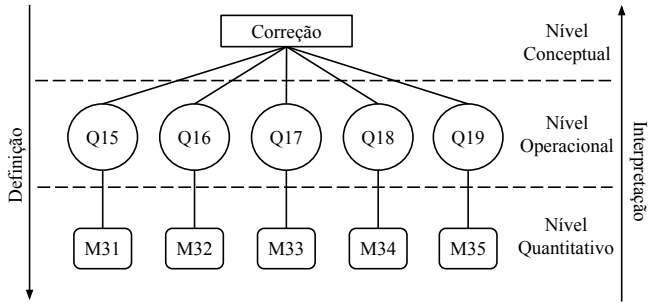


Figura 4.7: Aplicação da Abordagem GQM - Correção

Para a avaliação da qualidade dos modelos de requisitos *i**, foram identificados 3 (três) objetivos de medição específicos, 19 (dezanove) questões que caracterizam a forma como os objetivos são obtidos, e 35 (trinta e cinco) métricas principais que fornecem a informação quantitativa necessária para responder às questões definidas. Foram ainda definidas 80 (oitenta) métricas auxiliares, que podem ser consultadas no anexo C.

4.5 Sumário

Neste capítulo foi vista a aplicabilidade do modelo GQM para a avaliação da qualidade de modelos de requisitos *i**. Seguindo esta abordagem, foi apresentado o conjunto de métricas propostas para a avaliação de modelos *i** quanto à sua complexidade, completude e correção. Cada métrica foi explicada e definida, tendo também sido fornecida informação sobre métricas auxiliares necessárias.

Adicionalmente, foi fornecida informação sobre o meta-modelo do *framework i**, necessário para compreender a especificação formal das métricas em OCL e para a sua correta recolha de forma automática.

Por forma a se ter uma visão geral dos objetivos, questões e métricas, foi ainda apresentada a hierarquia completa da definição das métricas, com a apresentação dos níveis conceptual, operacional e quantitativo, de acordo com a abordagem GQM.

5

Implementação do Protótipo

Este capítulo apresenta as alternativas de implementação que foram consideradas, seguidas de informação sobre as tecnologias utilizadas no desenvolvimento da ferramenta que permite a criação de modelos *i** e torna automático o processo de recolha de métricas. Depois, é apresentada informação em relação à forma como a ferramenta lida com prevenção e detenção de erros. Em seguida, é apresentada informação relativa ao suporte à interoperabilidade, ou seja, a importação de um modelo feito com outra ferramenta para o protótipo, e a extração das métricas da ferramenta para um ficheiro de texto. Por fim, é apresentado um cenário de utilização, onde se pode observar a ferramenta em execução.

5.1 Alternativas de Implementação

Para a implementação do protótipo, foram consideradas algumas alternativas. Uma forma de se construir o protótipo, seria realizar o seu desenvolvimento de raiz, com recurso a linguagens de programação de alto nível, sendo necessário proceder-se à implementação de uma interface gráfica que permitisse a criação de modelos *i**. No entanto, tal processo seria moroso, e a automatização da recolha de métricas seria um desafio.

Existindo diversas ferramentas disponíveis para a construção de modelos *i**, uma outra opção seria a extensão de uma dessas ferramentas, por forma a incluir a definição e recolha automática de métricas. No entanto, seria necessário proceder-se à escolha de uma das ferramentas e estudar a sua implementação, o que também seria um processo moroso. Adicionalmente, as ferramentas disponíveis capturam parcialmente a linguagem *i**, não suportando todos os seus elementos, o que faz com que não seja possível recolher algumas das métricas definidas. Para além disso, as ferramentas apenas verificam parcialmente os erros cometidos pelos utilizadores, o que potencia a construção de

modelos incorretos.

Uma terceira opção prende-se com a utilização de técnicas de construção de linguagens para domínio específico. Estas técnicas permitem aumentar a eficiência do processo de desenvolvimento, sem recorrer a linguagens genéricas complexas, nem perder a flexibilidade de especificação. Adicionalmente, oferecem um nível de abstração adequado para o utilizador final. Desta forma, o desenvolvimento da ferramenta torna-se mais eficiente e menos moroso, permitindo criar suporte não só para a criação e edição de modelos *i**, mas também para a sua avaliação, através de regras de boa formação dos modelos, e torna automático o processo de recolha de métricas. Havendo conhecimento prévio sobre como usar estas técnicas, o tempo necessário para o seu estudo também fica minimizado. Assim, optou-se pela utilização de **técnicas de construção de linguagens para domínio de específico**, tomando partido das tecnologias existentes para o IDE Eclipse.

5.2 Tecnologias Utilizadas

Para a construção da ferramenta, utilizou-se o IDE Eclipse Modeling Tools [Ecl14c] com as tecnologias EMF [Ecl14b], GMF [Ecl14a], Emfatic [Ecl14d] e Epsilon [Ecl14e], tendo-se como ponto de partida a LDE i* [Nun09; Mon10]. A LDE i* foi criada com o objetivo de se apresentar uma ferramenta para a construção de modelos *i**, com mecanismos que permitem gerir a complexidade e correção dos mesmos. Para a criação da linguagem, foram utilizados os *plugins* EMF e GMF do IDE Eclipse. Sendo que não foi possível integrar corretamente esse projeto, devido ao facto de terem ocorrido atualizações significativas nos *plugins* EMF e GMF, a ferramenta criada para a presente dissertação foi **implementada de raiz**.

O EMF e GMF são *plugins* que permitem implementar e desenhar a ferramenta. Através do EMF é possível construir um modelo Ecore, que consiste no meta-modelo de uma linguagem para domínio específico. Não sendo possível expressar todas as regras sintáticas de uma linguagem através do seu meta-modelo, foram adicionadas restrições através da linguagem OCL, utilizando o editor OCLinEcore [Ecl14k]. O meta-modelo também pode ser definido através da linguagem textual Emfatic [Ecl14d], sendo que do ficheiro Ecore é possível gerar o ficheiro Emfatic e vice-versa. Assim sendo, qualquer modificação feita em um desses ficheiros é facilmente transposta para o outro. A linguagem Emfatic foi utilizada para definir o aspeto visual dos elementos presentes no *framework i**, assim como para definir formalmente as métricas. Para a construção do meta-modelo, teve-se em conta os apresentados em [Aya+05; Luc+08; Ale+08; Nun09], tendo sido efetuadas algumas alterações para melhor satisfazer as necessidades, com explicitado no capítulo anterior (capítulo 4).

O GMF utiliza os elementos do meta-modelo (tanto Ecore como Emfatic), fornecido pelo EMF, e cria um editor gráfico. A figura 5.1 apresenta o fluxo de desenvolvimento do editor gráfico, sendo que este é gerado pelo GMF *dashboard*, uma perspetiva dos projetos EMF/GMF.

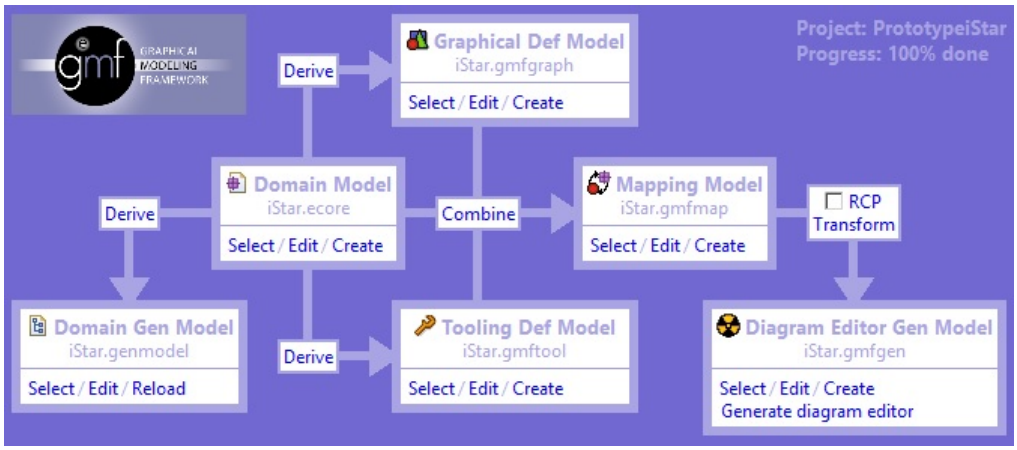


Figura 5.1: Fluxo de desenvolvimento do editor gráfico

Os ficheiros *gmfgraph*, *gmftool* e *gmfmap* são gerados automaticamente, com o auxílio da ferramenta EuGENia [Ecl14i], através dos dados presentes no ficheiro Ecore, e podem ser editados manualmente ou através da linguagem EOL (do inglês *Epsilon Object Language*) [Ecl14g]. O EOL é uma linguagem de programação imperativa para a criação e modificação de modelos EMF, e foi utilizada para complementar as formatações presentes no ficheiro Emfatic, assim como para definição de novos elementos decorativos e manipulação de alguns aspetos dos ficheiros *gmfgraph*, *gmftool* e *gmfmap*. Também foi utilizada para a definição da forma como se apresentam os valores das diferentes métricas.

O editor gráfico GMF permite a validação dos elementos nele desenhados. Esta validação tanto pode ser feita a nível do Ecore, como através da linguagem EVL (do inglês *Epsilon Validation Language*) [Ecl14h], que trabalha sobre a linguagem EOL e que permite a especificação de restrições, sendo bastante semelhante à linguagem OCL. A partir do momento em que a validação do editor gráfico é ativada, este reconhece automaticamente ficheiros *evl*. A validação permite verificar a correção do modelo, através da utilização de algumas das métricas definidas no ficheiro Emfatic. Assim, é possível verificar erros e situações não aconselhadas e fornecer informação ao utilizador. Para casos de erro, são definidas restrições através da utilização de `constraint`. Para situações que não são recomendadas, mas que não sejam consideradas erradas, utiliza-se `critique`. Em qualquer um dos casos, é apresentada uma mensagem informativa do erro ou sugestão.

No anexo B é feita uma explicação de como o projeto responsável pela criação da ferramenta pode ser importado para o IDE Eclipse, quais os *plugins* necessários, e quais as versões dos mesmos.

5.3 Prevenção e Deteção de Erros

Para a verificação da correção dos modelos criados com a ferramenta, há duas medidas que devem ser tomadas: a prevenção e a deteção de erros.

Como referido anteriormente, a prevenção de erros impede o utilizador de cometer

determinados erros do ponto de vista da modelação, não permitindo que este realize certas ações. A prevenção de erros é feita em tempo real, através de restrições apresentadas no modelo Ecore. Como se pode observar pelo meta-modelo apresentado no anexo A, um exemplo de restrição é o facto de uma ligação de associação *Plays* apenas poder ser utilizada entre um agente e um papel. Assim, o utilizador não tem a possibilidade de cometer erros nas associações, uma vez que não são permitidas associações incorretas.

Quando se está a modelar, existem alguns erros devido ao facto de ainda não terem sido modelados todos os elementos e relações necessários. Neste casos, não se deve impedir a colocação dos elementos ou ligações pretendidos, mas o erro deve ser detetado quando o utilizador pedir a verificação da correção do modelo. Assim, a deteção de erros permite que o utilizador realize a ação que deseja, mas informa-o de que o modelo não está correto. A deteção é feita posteriormente, a pedido do utilizador, e mostra tanto o erro, como uma pequena descrição do mesmo e sugestões sobre como o corrigir. Esta deteção é feita com o auxílio da linguagem EVL, e a listagem 5.1 apresenta um exemplo de código nessa linguagem.

Listagem 5.1: Exemplo de EVL para deteção de erros

```
context ISTAR {
  constraint hasElement {
    check: self.NElem() > 0
    message: "O modelo deve ter pelo menos um elemento."
    fix {
      title: "Adicione um elemento ao modelo."
    }
  }
}
```

5.4 Suporte à Interoperabilidade

O suporte à interoperabilidade do protótipo é feito de 2 (duas) formas distintas. A primeira prende-se com a importação de um modelo, construído com outra ferramenta, para o protótipo. A segunda está relacionada com a exportação das métricas da ferramenta para um ficheiro de texto, por forma a ser possível uma análise mais detalhada em ferramentas de análise de dados.

5.4.1 Importação de Modelos

A ferramenta foi construída por forma a ser possível criar modelos de requisitos *i** a partir do zero. No entanto, tendo em conta que os casos de estudo a serem utilizados nesta dissertação já se encontram modelados, torna-se útil a possibilidade de importar modelos criados anteriormente para a ferramenta. Desta forma, o tempo de modelação é reduzido de forma significativa, e diminuem-se os erros introduzidos por cópia manual do modelo.

Tendo em conta que a ferramenta de construção de modelos de requisitos *i** mais utilizada é a OpenOME, optou-se por criar um importador de modelos para os que são construídos com esta ferramenta. Um modelo criado com a ferramenta OpenOME é constituído por um ficheiro XML, com a extensão `.xmi`, que contém toda a sua especificação. A partir desse ficheiro, faz-se a geração de um ficheiro XML com a especificação do modelo criado com a ferramenta desenvolvida para a presente dissertação, através de EGL (do inglês *Epsilon Generation Language*) [Ros+08; Ecl14f]. O EGL é uma linguagem para geração de código, documentação ou outros artefactos textuais a partir de modelos, através da realização de uma transformação do tipo *model-to-text*. O ficheiro gerado pode ser importado diretamente na ferramenta, ficando pronto para que as métricas possam ser recolhidas.

5.4.2 Exportação de Métricas

As métricas, para além de serem definidas, calculadas e o seu resultado apresentado na ferramenta, necessitam de ser extraídas da mesma e escritas num ficheiro de texto, de forma a serem posteriormente analisadas e comparadas. A extração manual do resultado das métricas é um processo moroso e propenso a erros, uma vez que podem ser facilmente introduzidas incorreções, de forma involuntária, devido a faltas de atenção. Assim, torna-se importante a existência de uma forma automática de interpretar os modelos gerados pela ferramenta e retornar toda a informação necessária, tal como o resultado do cálculo das métricas e uma listagem dos elementos presentes no modelo. Através da informação obtida e guardada num ficheiro de texto, é então possível fazer uma avaliação cuidada das métricas definidas.

Um modelo criado com a ferramenta é constituído por um ficheiro XML, com a extensão `.xmi`, que contém toda a sua especificação. A partir desse ficheiro, faz-se a geração do ficheiro de texto onde os valores das métricas vão ser guardados, através de EGL. O ficheiro de texto gerado apresenta informação sobre o sistema, os elementos e ligações presentes no sistema, as métricas e os valores das métricas. Esse ficheiro tem o formato CSV (do inglês *Comma-separared Values*), por forma a facilitar a sua posterior utilização, aquando da avaliação dos resultados.

5.5 Cenário de Utilização

Por forma a se ter uma noção mais clara da ferramenta e de como esta funciona, apresenta-se um excerto do caso de estudo *Media Shop* (MS) [CKM01]. O seu principal objetivo é permitir que clientes analisem os artigos no catálogo *online* presente na Internet (livros, jornais, revistas, CD de áudio, cassetes de vídeo, entre outros) e realizem encomendas. A figura 5.2 mostra um fragmento do modelo MS modelado com a ferramenta, onde se apresenta o ator *Media Shop* e os elementos presentes na sua fronteira.

A tarefa ***Run Shop*** é decomposta em 5 (cinco) elementos, sendo eles o *softgoal* ***Improve***

Service, as sub-tarefas ***Manage Inventory*** e ***Manage Staff***, e os objetivos ***Handle Billing*** e ***Handle Customer Orders***. A tarefa ***Staff Training*** e os *softgoals* ***Satisfy Customer Desires*** e ***Be Friendly*** contribuem para se alcançar o atributo de qualidade especificado pelo *softgoal* ***Improve Service***. A tarefa ***Manage Inventory*** é decomposta em 2 (duas) sub-tarefas, sendo elas ***Sell Stock*** e ***Enhance Catalogue***. O objetivo ***Handle Customer Orders*** pode ser atingido através da tarefa ***Order In Person***, ***Order By Phone***, ou ***Order By Internet***. Por fim, a tarefa ***Order In Person*** é decomposta através das sub-tarefas ***Consult Catalogue***, ***Determine Amount*** e ***Select Items***.

No lado direito da ferramenta, é apresentada uma palete que contém todos os elementos e ligações necessários para a criação de modelos de requisitos *i**. Em baixo, podem ser observados os valores das diferentes métricas para o modelo em questão. Abaixo das métricas, e após se requerer a validação do modelo, são apresentados os seus problemas e aspetos não recomendados, assim como sugestões para a sua correção e melhoria.

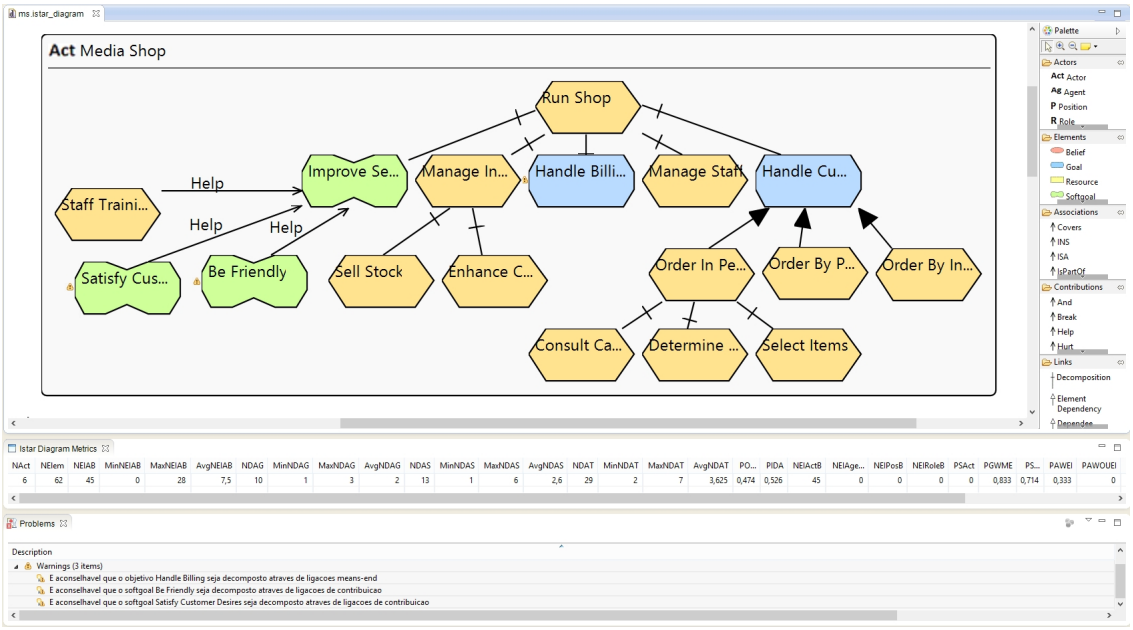


Figura 5.2: Aplicação da ferramenta ao caso de estudo *Media Shop*

5.6 Sumário

Neste capítulo apresentaram-se as alternativas de implementação que foram consideradas. Depois, foram apresentadas as principais tecnologias e linguagens utilizadas para o desenvolvimento do protótipo da ferramenta.

Foi visto como é feita a importação de um modelo criado com outra ferramenta para o protótipo. Também foi visto como é feita a extração das diferentes métricas calculadas pela ferramenta para um ficheiro de texto CSV, por forma a facilitar a sua posterior utilização e análise. Por fim, foi apresentado um cenário de utilização, onde se pode observar o aspeto gráfico da ferramenta, assim como os seus principais componentes.

6

Avaliação

Este capítulo contém toda a informação referente a avaliação das métricas definidas anteriormente, através da sua aplicação a diversos casos de estudo. Numa primeira parte, são apresentados os casos de estudo a serem utilizados. Depois, é apresentada informação sobre a adaptação dos modelos à norma do *framework i**. Em seguida, são apresentados os resultados da aplicação das métricas aos casos de estudo, para os objetivos complexidade, completude e correção. Por fim, é apresentada uma discussão crítica dos resultados obtidos, e é feita uma validação teórica das métricas, através de propriedades de Weyuker.

6.1 Casos de Estudo

Para a seleção dos casos de estudo a serem utilizados, tentou-se encontrar **sistemas de *software* reais** cuja modelação fosse feita através da linguagem *i**, e que não estivessem protegidos por sigilo industrial. No entanto, devido ao facto de esses modelos não serem em número significativo para uma análise o mais completa possível, em alguns casos os sistemas de *software* apresentados são de **origem académica**. Devido a tal facto, a dimensão dos sistemas de *software* estudados é variável e, consequentemente, a dimensão dos modelos também o será.

Cada caso de estudo apresenta uma breve descrição do seu propósito e dos seus principais objetivos, contendo uma referência para uma apresentação gráfica do seu modelo, os resultados das métricas, e informação sobre os erros detetados pela ferramenta.

6.1.1 Sistema *Media Shop*

O caso de estudo *Media Shop* (MS) [CKM01] representa uma loja que vende diferentes tipos de artigos, tais como livros, jornais, revistas, CD de áudio, cassetes de vídeo, entre outros. Os clientes da loja podem consultar um catálogo que descreve os artigos disponíveis, assim como realizar encomendas. O catálogo é periodicamente atualizado. Por forma a expandir a sua quota de mercado, a gerência da loja decidiu implementar vendas através da Internet. Assim, o principal objetivo do sistema é permitir que um cliente *online* analise os artigos presentes no catálogo e realize encomendas.

Este é um sistema bastante utilizado a nível académico como exemplo para propósitos distintos, tendo por isso existido várias modificações, feitas ao longo do tempo, para melhor se adequar às diferentes necessidades. Apesar de existirem várias versões, a versão utilizada é a original. De notar que a versão original do modelo do sistema não utiliza o *framework i** Yu'95 [Yu95], mas sim a variante Tropos [GMS05].

No anexo D, as figuras D.1 e D.2 apresentam informação resultante da modelação deste caso de estudo pela ferramenta.

6.1.2 Sistema *Newspaper Office*

O caso de estudo *Newspaper Office* (NO) [Sil+05] representa um jornal que possui uma página na Internet, sendo que os utilizadores da página do jornal pretendem ler notícias. As notícias relacionadas com um tema específico são extraídas de diferentes agências de notícias, que estão distribuídas pela Internet, são traduzidas e fundidas por forma a proporcionar notícias de boa qualidade. As notícias são escritas por jornalistas, que podem contactar fotógrafos caso pretendam auxiliares visuais para a informação descrita, e são aprovadas por um editor. O administrador, por sua vez, coloca as notícias na página na Internet, para que os utilizadores as possam ler.

Este sistema é utilizado a nível académico como exemplo para propósitos distintos, nomeadamente como auxiliar no ensino da linguagem *i**. De notar que o modelo original do sistema não utiliza o *framework i** Yu'95, mas sim a variante Tropos.

No anexo D, as figuras D.3 e D.4 apresentam informação resultante da modelação deste caso de estudo pela ferramenta.

6.1.3 Sistema *Health Care*

O caso de estudo *Health Care* (HC) [Yu95] representa um sistema de assistência médica, em que os custos médicos referentes ao tratamento de um determinado paciente são suportados por uma companhia de seguros, a troco de um pagamento anual de um montante definido anteriormente. Por forma a que o médico receba o seu pagamento através da companhia de seguros, o tratamento em si tem que ser previamente aprovado. É, ainda, necessário verificar se o seguro do paciente é aplicável, tendo em conta o preço da consulta e o tratamento prescrito pelo médico.

Este sistema foi proposto como parte de uma tese de doutoramento, por forma a se explicar o funcionamento do *framework i**.

No anexo D, as figuras D.5 e D.6 apresentam informação resultante da modelação deste caso de estudo pela ferramenta.

6.1.4 Sistema *Health Protection Agency*

O caso de estudo *Health Protection Agency* (HPA) [Eng09; Loc+12] representa o sistema de vigilância de pacientes infetados com o vírus VIH/SIDA (HAP, do inglês *HIV/AIDS Prevention*), da Agência de Proteção da Saúde do Reino Unido [Hea14].

Este sistema representa um caso real. O sistema de vigilância HAP faz a monitorização do impacto do vírus VIH sobre a saúde pública no Reino Unido, através da execução de um sistema de *software* projetado originalmente em Setembro de 1982. O sistema evoluiu ao longo dos anos, sendo agora considerado como um dos sistemas mais abrangentes de vigilância de saúde pública, cobrindo os principais componentes recomendados pelo Centro de Controlo e Prevenção de Doenças (CDC, do inglês *Center for Disease Control and Prevention*) [Cen14; CG14].

No anexo D, as figuras D.7 e D.8 apresentam informação resultante da modelação deste caso de estudo pela ferramenta.

6.1.5 Sistema *National Air Traffic Services*

O caso de estudo *National Air Traffic Services* (NATS) [Loc+12] representa o sistema de controlo de tráfego aéreo do Reino Unido [Nat14].

Este sistema representa um caso real. O espaço aéreo do Reino Unido está dividido em dois tipos: controlado e não controlado. A tripulação de determinado avião deve obter autorização do controlo de tráfego antes de entrar em espaço aéreo controlado. No entanto, os pilotos nem sempre estão cientes da sua localização e, antes da construção de um novo modelo do sistema, o número de aeronaves que entravam no espaço aéreo controlado sem autorização para o fazer tinha vindo a aumentar. Os controladores apercebiam-se que uma aeronave desconhecida entrava no espaço aéreo controlado quando observavam o monitor do radar. O novo sistema pretende implementar uma ferramenta de segurança que forneça aos controladores avisos atempados de violação do espaço aéreo controlado, de forma a minimizar os efeitos da transgressão.

No anexo D, as figuras D.9 e D.10 apresentam informação resultante da modelação deste caso de estudo pela ferramenta.

6.1.6 Sistema *My Courses*

O caso de estudo *My Courses* (MC) [Lim+11] representa um sistema de agendamento de cursos de uma universidade. O sistema permite verificar disponibilidade de salas de aula e de professores, assim como quais os alunos matriculados em determinado curso. Adicionalmente, calcula e propõe um cronograma de cursos, gere dados a respeito de

utilizadores do sistema, programas curriculares, cursos e os seus respetivos recursos, e permite a comunicação de informação sobre alterações efetuadas nos cursos.

Este sistema foi especificado por estudantes da Universidade Federal de Pernambuco (Brasil), para o Score Contest de 2011 [SCO14], uma competição mundial em engenharia de *software*, destinada a estudantes de licenciatura e mestrado.

No anexo D, as figuras D.11 e D.12 apresentam informação resultante da modelação deste caso de estudo pela ferramenta.

6.1.7 Sistema *Mobile Media*

O caso de estudo *Mobile Media* (MM) [Bor09] representa um sistema de manipulação de diferentes tipos de médias digitais, como fotografias, músicas e vídeos, em dispositivos móveis, nomeadamente telemóveis. É possível adicionar e remover os vários média, assim como fazer a sua gestão, criando álbuns e adicionando legendas. Adicionalmente, também se pode atualizar a lista de médias que se encontram no dispositivo móvel através de serviços de alojamento e partilha presentes na Internet.

Este sistema teve por base uma linha de produtos de *software* (SPL, do inglês *Software Product Lines*) [You05; Fig+08], tendo sido especificado como parte de uma tese de mestrado, por forma a mostrar como a abordagem G2FM (do inglês *Goal To Feature Model*) pode ser utilizada para se extrair *features* de um modelo *i**.

No anexo D, as figuras D.13 e D.14 apresentam informação resultante da modelação deste caso de estudo pela ferramenta.

6.1.8 Sistema *By The Way*

O caso de estudo *By The Way* (BTW) [BHX09] representa um sistema de planeamento de rotas que permite entrada de dados por parte da comunidade. Quando uma rota é definida, o utilizador pode receber conselhos sobre a rota, fornecidos por outros utilizadores, por forma a ajudar no seu planeamento. Existe a possibilidade de os conselhos terem que ser filtrados, de maneira a ser fornecida informação relevante ao utilizador sobre o local que este pretende visitar.

Este sistema foi especificado por estudantes da Universidade Federal de Pernambuco (Brasil), para o Score Contest de 2009.

No anexo D, as figuras D.15 e D.16 apresentam informação resultante da modelação deste caso de estudo pela ferramenta.

6.1.9 Sistema *Meeting Scheduler*

O caso de estudo *Meeting Scheduler* (MSr) [Yu95] representa um sistema para agendamento de reuniões. Para cada pedido de reunião, o sistema tenta determinar uma data e uma localização que sejam adequadas para a maioria dos participantes, após recolher informação sobre a sua disponibilidade. A disponibilidade é composta por datas em que o participante não está disponível, e por um conjunto de preferências. A data escolhida

não deve ser uma na qual os participantes não estão disponíveis, e deve estar dentro do máximo de preferências possível. Por fim, os participantes devem concordar com a data, assim que uma data aceitável seja encontrada.

Este sistema foi proposto como parte de uma tese de doutoramento, por forma a se explicar o funcionamento do *framework i**.

No anexo D, as figuras D.17 e D.18 apresentam informação resultante da modelação deste caso de estudo pela ferramenta.

6.1.10 Sistema *Patient Wellness Tracking*

O caso de estudo *Patient Wellness Tracking* (PWT) [An+09] representa um sistema de acompanhamento do bem-estar de um paciente num centro de saúde gerido por profissionais de enfermagem. O centro de saúde utiliza uma abordagem de cuidados interdisciplinar para gerir as doenças. O sistema tem como objetivo manter e gerir informação sobre uma grande variedade de serviços de saúde e bem-estar prestados aos pacientes.

Este sistema foi proposto como caso de estudo para a adequação da utilização do *framework i** na elicitação e modelação de requisitos no contexto de tecnologias de informação para a área da saúde.

No anexo D, as figuras D.19 e D.20 apresentam informação resultante da modelação deste caso de estudo pela ferramenta.

6.2 Adaptação dos Casos de Estudo à Norma *i**

Os casos de estudo descritos anteriormente foram modelados através da ferramenta criada para esta dissertação. Como referido na secção 5.3, a ferramenta foi criada por forma a evitar, sempre que possível, a construção de modelos incorretos, quando estes são criados de raiz, tendo sido implementadas restrições por forma a não se cometerem erros de modelação segundo os guias apresentados na página *wiki* do *i**. Assim, casos de estudo que estejam a violar alguma regra de modelação imposta pela ferramenta, foram alterados por forma a permitir a sua construção. No entanto, de maneira a que algumas incorreções pudessem ser detetadas e quantificadas para o objetivo correção, teve que se recorrer aos modelos originais e não aos modelos modificados. Ao utilizar os modelos originais, as métricas de correção não podem ser calculadas de modo automático, mas o nível de correção dos diferentes modelos é mais preciso.

Devido ao facto de alguns modelos não seguirem o *framework i** Yu'95 mas sim outras variantes, e de a ferramenta ter sido construída tendo por base a variante Yu'95, os modelos também tiveram que ser adaptados em conformidade. As principais diferenças prendem-se com a nomenclatura das ligações de contribuição. Um exemplo é a utilização, por parte do modelo, da ligação de contribuição +, tendo sido substituída por *Some+*.

No anexo E podem ser verificadas quais as diferenças entre o caso de estudo original e a sua modelação com a ferramenta.

6.3 Resultados das Métricas

Os resultados das métricas foram separados pelos objetivos complexidade, completude e correção. Dentro das secções correspondentes a cada um dos objetivos, são apresentados gráficos de colunas e gráficos *boxplot*. Estes últimos permitem mostrar o grau de dispersão e assimetria dos dados obtidos. A figura 6.1 apresenta os principais conceitos referentes aos gráficos *boxplot*, sendo que `A` representa a altura da caixa. De notar que os valores atípicos e extremos estão representados apenas para valores superiores, mas também podem existir para valores inferiores da amostra.

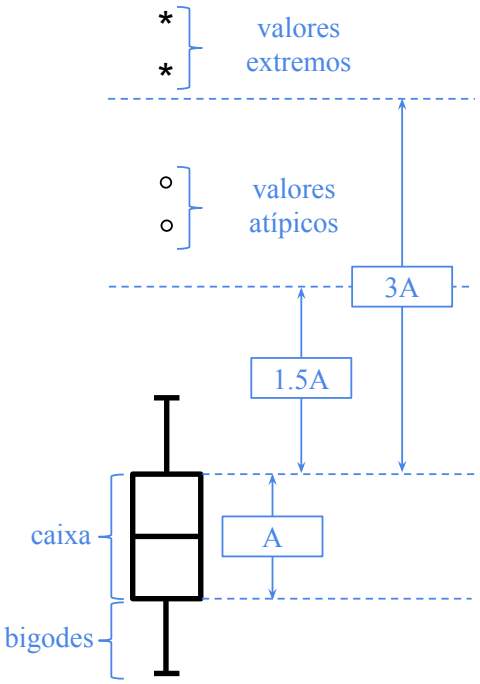


Figura 6.1: Gráfico *boxplot*, com representação de valores atípicos e extremos

Os gráficos *boxplot* são compostos por 3 (três) elementos principais:

- **Caixa** – contém os valores entre os percentis 25 e 75 da amostra, apresentando 3 (três) estatísticas, sendo elas:
  - **Quartil inferior** – corresponde ao limite inferior da caixa;
  - **Mediana** – corresponde ao centro da amostra;
  - **Quartil superior** – corresponde ao limite superior da caixa.
- **Bigodes** – estendem em 1,5 vezes o comprimento da caixa ou, no caso de nenhum elemento da amostra se encontrar fora desses limites, indicam os valores mínimos e máximos da amostra. Numa amostra considerada normal, cerca de 95% dos dados encontram-se entre os bigodes;

- **Valores atípicos e extremos** – correspondem aos valores da amostra que se encontram desfasados dos restantes, tendo diferentes distâncias:

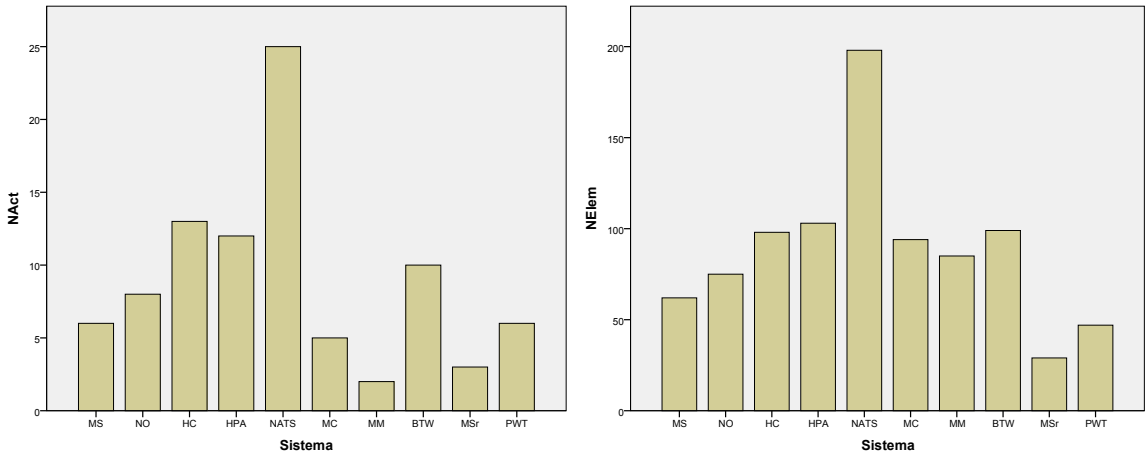
  - **Valores atípicos** – encontram-se à distância de uma caixa e meia do quartil inferior ou do quartil superior e são representados por ∘ (círculo);
  - **Valores extremos** – encontram-se à distância de três caixas do quartil inferior ou do quartil superior e são representados por ∗ (asterisco).

6.3.1 Resultados para o Objetivo Complexidade

Questão 1 Quão complexo é o modelo, em relação ao número de atores e elementos?

Na figura 6.2 é apresentado o número de atores (6.2a) e o número de elementos (6.2b), de cada caso de estudo.

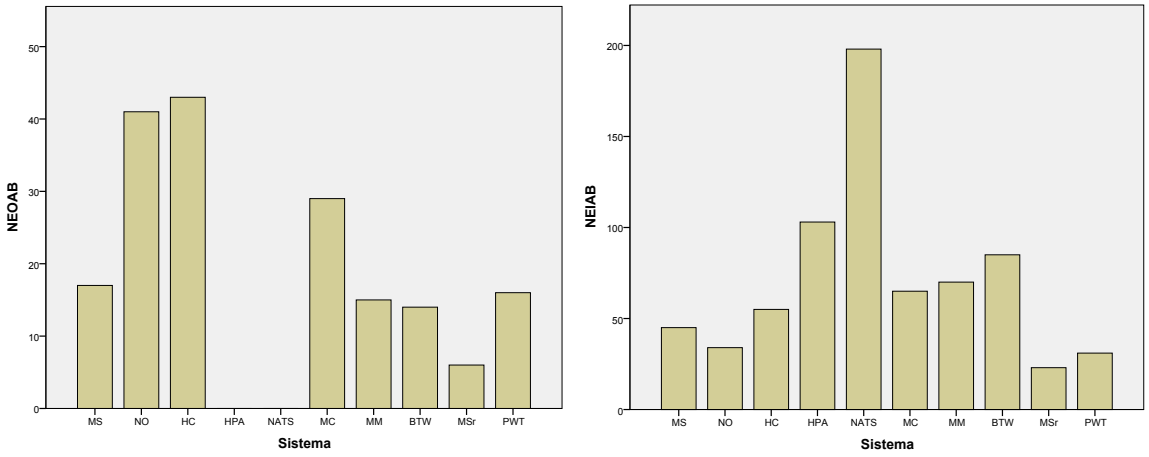
A nível de número de atores presentes no modelo, o sistema NATS tem, aproximadamente, o dobro do tamanho do segundo e terceiro maiores sistemas. O mesmo é verificável a nível do número de elementos, o que sugere que o sistema NATS é bastante mais complexo do que todos os outros sistemas. A complexidade de um sistema pode afetar a sua compreensão e, consequentemente, levar a erros, incongruências e ambiguidades relativamente à modelação, que se propagam para as fases seguintes do desenvolvimento do sistema de *software*. Assim, o sistema NATS deve ser analisado em mais detalhe, por forma a verificar se a sua complexidade é essencial ou acidental. O sistema MM é o que apresenta o número de atores mais reduzido, mas o seu número de elementos é similar ao de os sistemas com mais atores a seguir ao NATS, como HC, HPA, MC e BTW, o que pode sugerir que alguns dos seus atores têm muitas responsabilidades no modelo. No entanto, é necessário uma análise mais aprofundada.



(a) Número de atores (b) Número de elementos

Figura 6.2: Número de atores e elementos

Para uma análise mais aprofundada, é necessário verificar se existem diferenças em relação ao número de elementos externos e internos presentes nos modelos. Uma vez que todos os sistemas apresentam elementos dentro de fronteiras de atores, cada um deles é representado pelo modelo SR do *framework i**. O sistemas HPA e NATS não apresentam nenhum elemento externo (6.3a), mas têm os números mais elevados no que se refere a elementos internos (6.3b), o que sugere que todas as dependências estão ligadas diretamente aos elementos internos dos atores. Excetuando o sistema NO, todos os sistemas apresentam mais elementos internos do que externos, o que sugere que o nível de detalhe dos atores é elevado e revela um padrão de modelação. O sistema MM apresenta uma distribuição equilibrada entre os dois tipos de elementos.



(a) Número total elementos externos

(b) Número total de elementos internos

Figura 6.3: Número total de elementos fora e dentro das fronteira dos atores

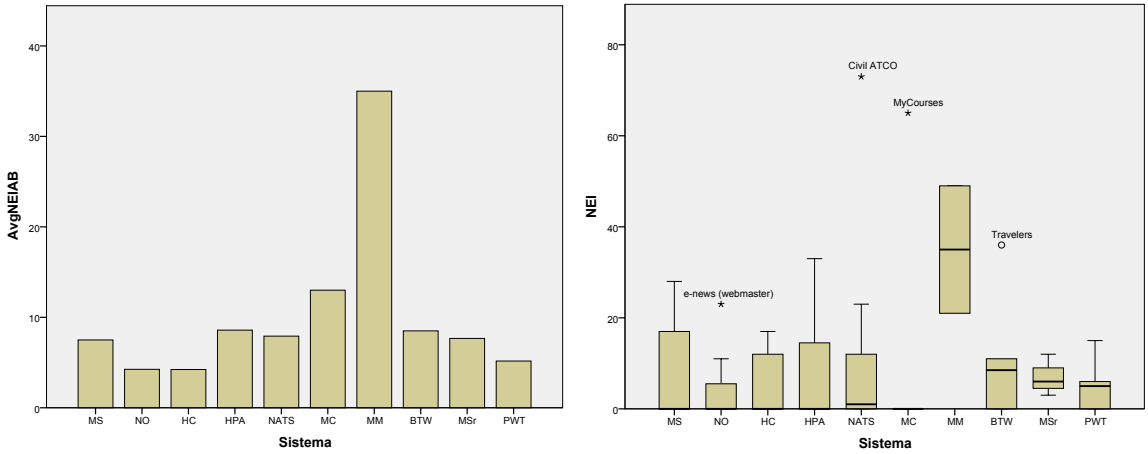
Questão 2 Um ator tem demasiada responsabilidade no modelo?

Na figura 6.4 é apresentado o número médio de elementos dentro da fronteira dos atores (6.4a) e o número de elementos internos em cada um dos atores (6.4b), de cada caso de estudo.

O número médio de elementos dentro da fronteira dos atores fornece informação sobre o rácio elementos/atores. O sistema MM apresenta maior rácio do que todos os outros sistemas, que apresentam rácios similares entre si. Este valor justifica-se pelo facto de o sistema possuir uma diferença significativa entre o número de elementos e o número de atores. O sistema mais complexo a nível de tamanho, NATS, tem um rácio com baixa densidade, o que sugere uma boa modularidade no geral.

Excetuando os sistemas MM e MSr, o número mínimo de elementos dentro da fronteira de um ator é 0 (zero), ou seja, existe pelo menos um ator cuja responsabilidade não foi atribuída, o que sugere que o ator em questão pode não ter uma importância

significativa no sistema. Apesar de o sistema MM ter um rácio elementos/atores elevado, não apresenta valores extremos ou atípicos, o que sugere que a atribuição de responsabilidades a atores é equilibrada. Os atores *e-news (webmaster)*, *Civil ATCO* e *MyCourses* são considerados extremos, e o ator *Travelers* é considerado atípico, o que sugere que têm demasiada responsabilidade no sistema. Estes atores seriam possíveis candidatos para uma futura decomposição em sub-atores, utilizando a associação *is-part-of*, onde cada sub-ator seria responsável por um sub-sistema. No entanto, tal é meramente indicativo, pelo que os atores devem ser analisados em mais detalhe (ver questões 3-5).



(a) Número médio de elementos internos

(b) Número de elementos internos aos atores

Figura 6.4: Média e número de elementos dentro das fronteiras dos atores

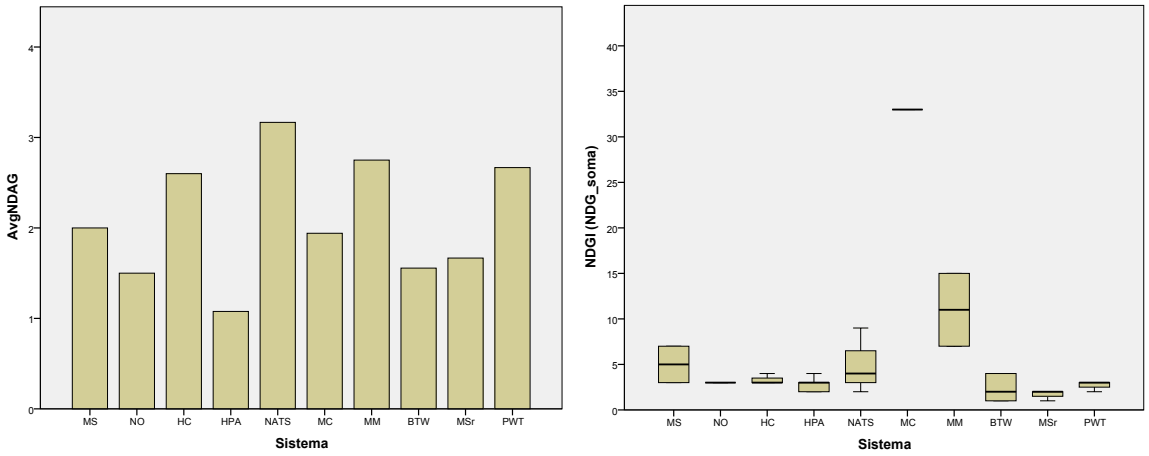
Questão 3 Quão complexo é um objetivo de um ator, em relação às suas decomposições?

Na figura 6.5 é apresentado o número médio de decomposições associadas a objetivos (6.5a) e o número de decomposições associadas a objetivos de atores (6.5b), de cada caso de estudo.

As decomposições dos objetivos permitem compreender como é que um objetivo é alcançado, ou seja, qual o meio necessário para atingir determinado fim. Existe uma variação no número médio de decomposições de objetivos de cada um dos sistemas, embora essa variação não seja muito significativa. O sistema HPA é o que apresenta o número médio mais baixo, com o valor 1 (um), o que sugere que os seus objetivos não são complexos, sendo facilmente compreensível como é que determinado objetivo é alcançado. Nos sistemas em que o número médio é mais elevado, como é o caso dos sistemas HC, NATS, MM e PWT, a compreensão de como determinado objetivo é alcançado pode não ser afetada, uma vez que a média, apesar de mais elevada, é relativamente baixa, não chegando a 4 (quatro).

Analisando o número total de decomposições de objetivos dentro de cada um dos atores, observa-se que não existem valores extremos nem atípicos em nenhum dos

sistemas. Tal facto sugere que o nível de complexidade dos objetivos, em relação às suas decomposições, é semelhante entre os diferentes atores do sistema.



(a) Média de decomposições de objetivos (b) Número total de decomposições de objetivos

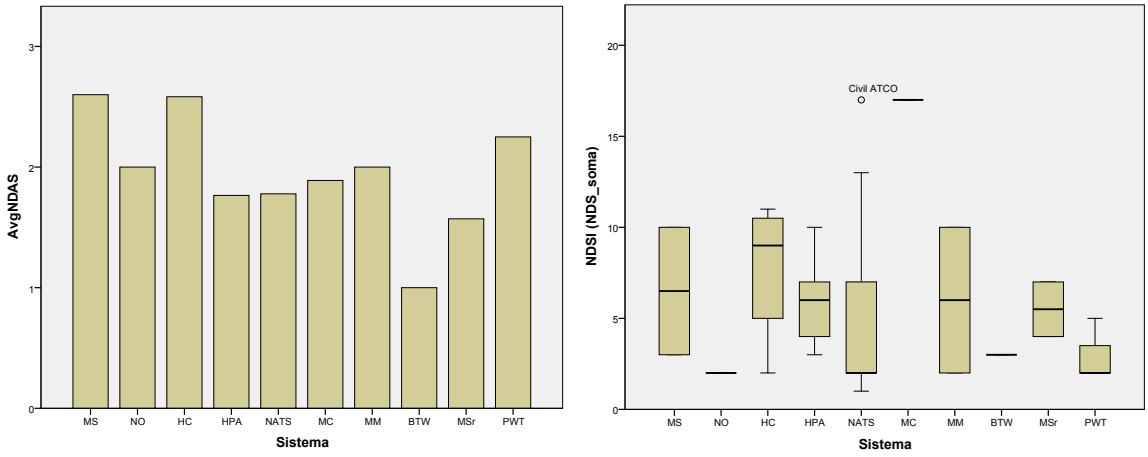
Figura 6.5: Média e número de decomposições associadas a objetivos

Questão 4 Quão complexo é um *softgoal* de um ator, em relação às suas decomposições?

Na figura 6.6 é apresentado o número médio de decomposições associadas a *softgoals* (6.6a) e o número de decomposições associadas a *softgoals* de atores (6.6b), de cada caso de estudo.

As decomposições dos *softgoals* permitem compreender como é que um *softgoal* é alcançado, ou seja, quais os elementos que contribuem para a sua realização. Existe uma variação no número médio de decomposições de *softgoals* de cada um dos sistemas, embora essa variação não seja muito significativa. O sistema HPA é o que apresenta o número médio mais baixo, com o valor 1 (um), o que sugere que os seus *softgoals* não são complexos, sendo facilmente compreensível como é que determinado *softgoal* é realizado. Nos sistemas em que o número médio é mais elevado, como é o caso dos sistemas HC, NATS, MM e PWT, a compreensão de como determinado *softgoal* é realizado pode não ser afetada, uma vez que a média, apesar de mais elevada, é relativamente baixa, não chegando a 3 (três).

Analisando o número total de decomposições de *softgoals* dentro de cada um dos atores, observa-se que, com exceção do sistema NATS, não existem valores extremos nem atípicos em nenhum dos sistemas. Tal facto sugere que o nível de complexidade dos *softgoals*, em relação às suas decomposições, é semelhante entre os diferentes atores do sistema. No sistema NATS, o ator *Civil ATCO* é apresentado como atípico, ou seja, possui *softgoals* cujas decomposições são mais complexas do que a média. Um nível de complexidade mais elevado pode afetar a compreensão de como determinado *softgoal* é realizado e, consequentemente, afetar a sua realização. Assim, o ator *Civil ATCO* deve ser analisado em mais detalhe.



(a) Média de decomposições de *softgoals* (b) Número total de decomposições de *softgoals*

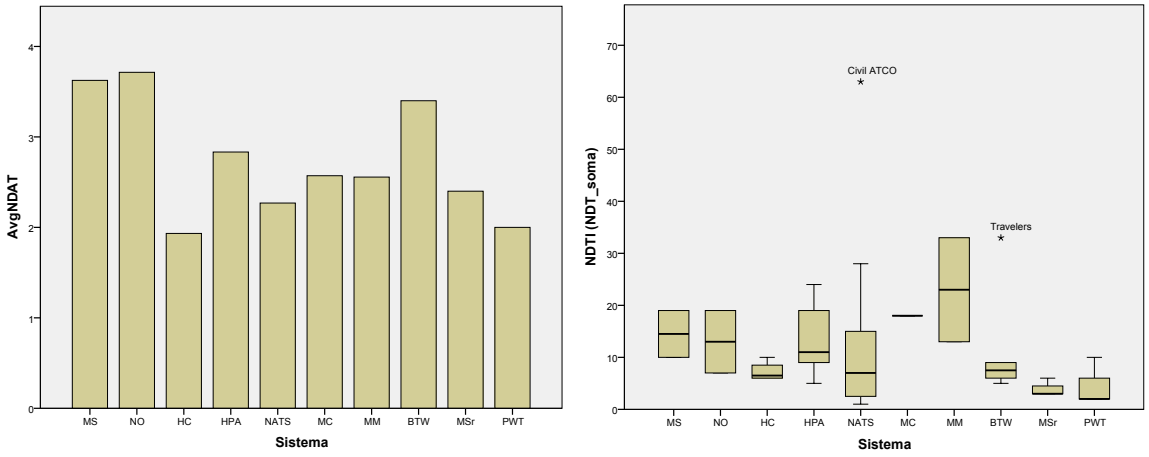
Figura 6.6: Média e número de decomposições associadas a *softgoals*

Questão 5 Quão complexa é uma tarefa de um ator, em relação às suas decomposições?

Na figura 6.7 é apresentado o número médio de decomposições associadas a uma tarefa (6.7a) e o número de decomposições associadas a tarefas de atores (6.7b), de cada caso de estudo.

As decomposições das tarefas permitem compreender como é que uma tarefa é alcançada, decompondo-a em sub-objetivos, sub-tarefas, recursos ou *softgoals*. Existe uma variação no número médio de decomposições de tarefas de cada um dos sistemas, embora essa variação não seja muito significativa. O sistema HC é o que apresenta o número médio mais baixo, com o valor 2 (dois), o que sugere que as suas tarefas não são complexas, sendo facilmente compreensível como é que determinada tarefa é alcançada. Nos sistemas em que o número médio é mais elevado, como é o caso dos sistemas MS, NO e BTW, a compreensão de como determinada tarefa é alcançada pode não ser afetada, uma vez que a média, apesar de mais elevada, é relativamente baixa, não chegando a 4 (quatro).

Analisando o número total de decomposições de tarefas dentro de cada um dos atores, observa-se que, com exceção dos sistemas NATS e BTW, não existem valores extremos nem atípicos nos sistemas. Tal facto sugere que o nível de complexidade das tarefas, em relação às suas decomposições, é semelhante entre os diferentes atores do sistema. No sistema NATS, o ator *Civil ATCO* é apresentado como extremo, ou seja, possui tarefas cujas decomposições são mais complexas do que a média. O mesmo acontece no sistema BTW, com o ator *Travelers*. Um nível de complexidade mais elevado pode afetar a compreensão de como determinada tarefa é alcançada. Assim, os atores *Civil ATCO* e *Travelers* devem ser analisados em mais detalhe.



(a) Média de decomposições de tarefas (b) Número total de decomposições de tarefas

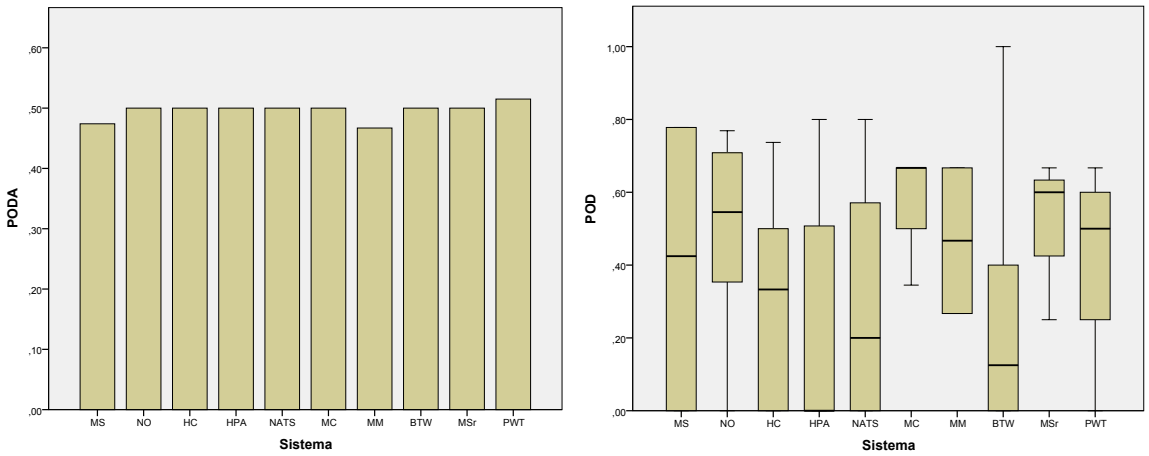
Figura 6.7: Média e número de decomposições associadas a tarefas

Questão 6 Um ator é demasiado dependente num modelo?

Na figura 6.8 é apresentada a percentagem total de dependências de saída (6.8a) e a percentagem de dependências de saída de atores (6.8b), de cada caso de estudo.

A percentagem de dependências de saída tem um valor próximo de 50% em todos os sistemas, o que sugere um equilíbrio entre as dependências de saída e de entrada. Este equilíbrio sugere que os atores não estão demasiado dependentes de outros.

Analisando a percentagem de dependências de saída associadas a atores, observa-se que não existem valores extremos nem atípicos nos sistemas. No entanto, o valor da mediana e dos bigodes é variável nos diferentes sistemas. O sistema que mais se destaca é o BTW, em que a mediana é relativamente baixa (cerca de 15%) mas o seu valor máximo é 100%, ou seja, existe um ator que apenas apresenta dependências de saída, sendo uma fonte.



(a) Percentagem de dependências de saída (b) Percentagem de dependências de saída/ator

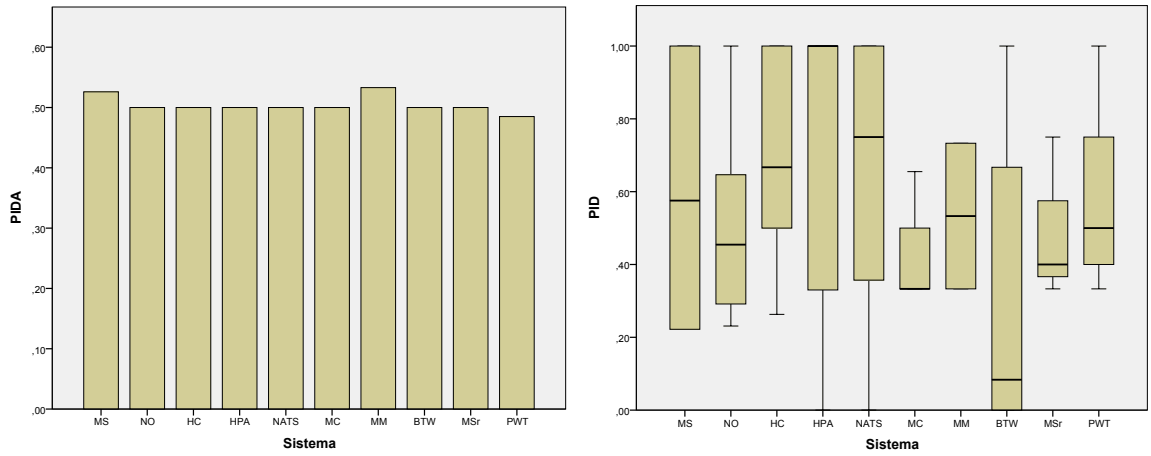
Figura 6.8: Percentagem de dependências de saída

Questão 7 Um ator tem demasiadas dependências num modelo?

Na figura 6.9 é apresentada a percentagem total de dependências de entrada (6.9a) e a percentagem de dependências de entrada de atores (6.9b), de cada caso de estudo.

A percentagem de dependências de entrada tem um valor próximo de 50% em todos os sistemas, o que sugere um equilíbrio entre as dependências de entrada e de saída. Este equilíbrio sugere que os atores não são indispensáveis aos outros.

Analisando a percentagem de dependências de entrada associadas a atores, observa-se que não existem valores extremos nem atípicos nos sistemas. No entanto, o valor da mediana e dos bigodes é variável nos diferentes sistemas. Com exceção dos sistemas MC, MM e MSr, todos os sistemas têm pelo menos um ator cujo valor máximo é 100%, ou seja, apresenta apenas dependências de entrada, sendo um poço.



(a) Percentagem de dependências de entrada (b) Percentagem de dep. de entrada/ator

Figura 6.9: Percentagem de dependências de entrada

Questão 8 Há variação na complexidade média dos diferentes tipos de atores?

Não existem atores com tipos específicos, pelo que não é possível verificar a variação na complexidade média dos diferentes tipos de atores (ver questão 9 da completude).

6.3.2 Resultados para o Objetivo Completude

Questão 9 Quão específicos são os atores?

Na figura 6.10 é apresentada a percentagem de atores com um tipo específico (agente, papel ou posição), de cada caso de estudo.

Como se pode observar, não existem atores específicos em nenhum dos sistemas, o que sugere um padrão de modelação. A falta de utilização de atores específicos pode fazer com que o modelo perca alguma informação útil, uma vez que a precisão e o nível de detalhe são menores.

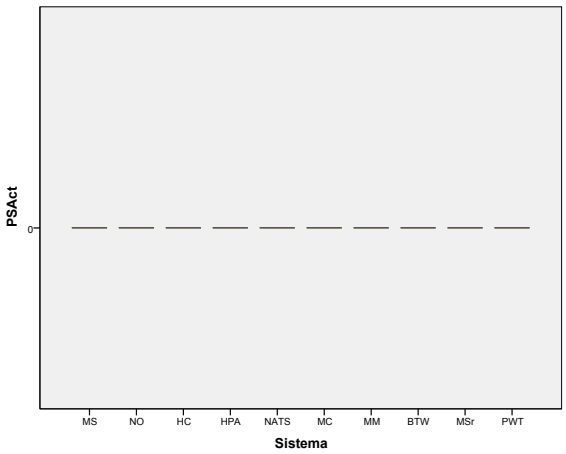


Figura 6.10: Percentagem de atores com um tipo específico

Questão 10 Quão detalhados são os objetivos?

Na figura 6.11 é apresentada a percentagem de objetivos com ligações *means-end*, de cada caso de estudo.

Tendo em conta que os objetivos são considerados elementos de alto nível, estes devem ser decompostos, de forma a não existir objetivos-folha no modelo. A percentagem de objetivos refinados é elevada, sendo que nos sistemas HC, MC, BTW e MSr o seu valor é de 100%, ou seja, todos os objetivos possuem ligações *means-end*.

Apesar de nenhum sistema ser considerado extremo ou atípico, e de a mediana ser elevada, alguns devem ser analisados em mais detalhe. É o caso dos sistemas NATS e MM, cujo valor da percentagem é inferior a 50%, sendo aconselhável verificar como é que os objetivos são satisfeitos e acrescentar a respetiva ligação *means-end*.

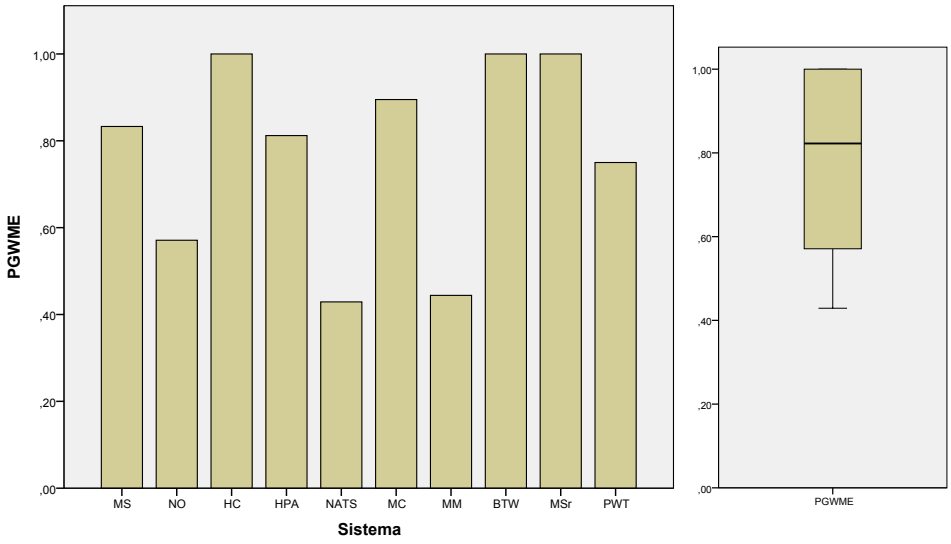


Figura 6.11: Percentagem de objetivos com *means-end*

Questão 11 Quão detalhados são os *softgoals*?

Na figura 6.12 é apresentada a percentagem de *softgoals* com ligações de contribuição, de cada caso de estudo.

Tendo em conta que os *softgoals* são considerados elementos de alto nível, estes devem ser decompostos e operacionalizados. A percentagem de *softgoals* refinados é elevada, sendo que nos sistemas MC, MM, MSr e PWT o seu valor é de 100%, ou seja, todos os *softgoals* possuem ligações de contribuição.

O sistema NO é apresentado como atípico, uma vez que o valor da percentagem é de cerca de 20%, um valor bastante inferior ao do apresentado nos restantes sistemas. Assim, este sistema deve ser analisado em mais detalhe, sendo aconselhável verificar como é que os *softgoals* são operacionalizados e acrescentar a respetiva ligação de contribuição.

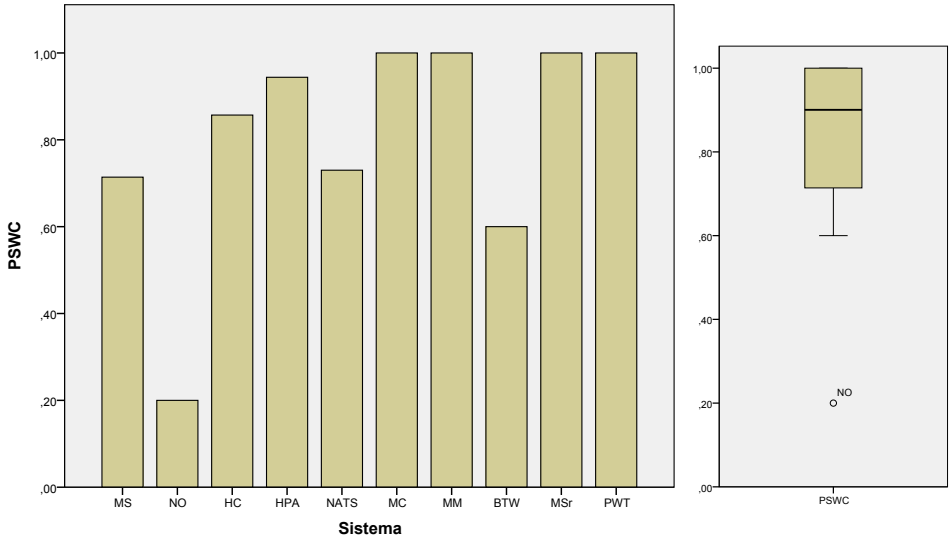


Figura 6.12: Percentagem de *softgoals* com contribuições

Questão 12 Quão detalhado é o modelo SR em relação aos seus atores?

Na figura 6.13 é apresentada a percentagem de atores com elementos presentes dentro da sua fronteira, de cada caso de estudo.

Os sistemas MM e MSr apresentam um modelo SR com um bom nível de detalhe, uma vez que o valor da percentagem de atores com elementos na sua fronteira é de 100%, ou seja, todos os atores estão detalhados.

Apesar de nenhum sistema ser considerado extremo ou atípico, a mediana encontra-se nos 50%, um valor relativamente baixo. No modelo SR, atores que não estejam suficientemente detalhados, ou seja, que não tenham elementos dentro da sua fronteira, são considerados incompletos e a sua presença no modelo pode não ser necessária. Os sistemas MS, NO, HC e MC devem ser analisados em mais detalhe,

por forma a verificar se alguns dos seus atores são mesmo necessários, ou se não oferecem nenhum tipo de informação relevante. Verificando-se o primeiro caso, é aconselhável que sejam adicionados elementos aos atores em questão, fazendo a atribuição das respetivas responsabilidades. Verificando-se o segundo caso, é aconselhável que os atores em questão sejam excluídos do modelo, uma vez que a sua presença não acrescenta algo de útil, e faz aumentar a complexidade do modelo como um todo.

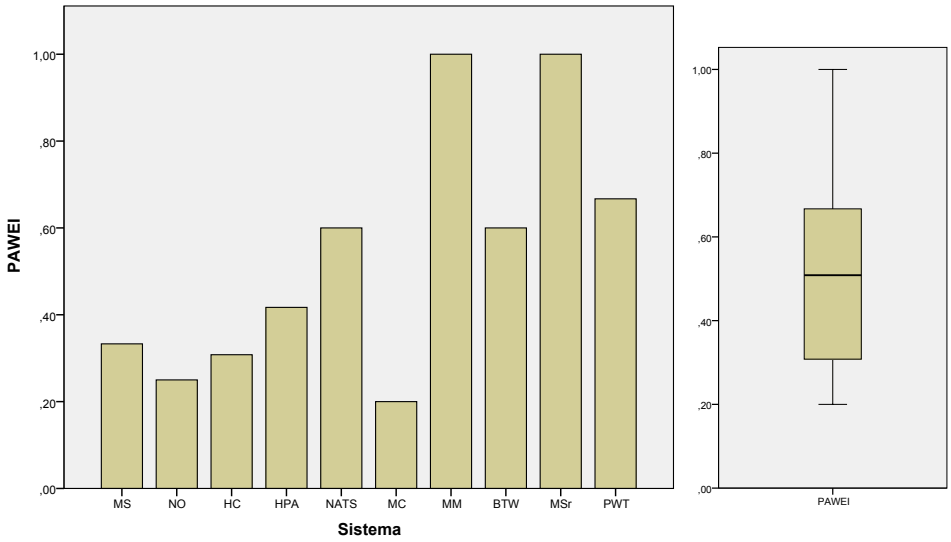


Figura 6.13: Percentagem de atores com elementos na sua fronteira

Questão 13 Quão perto estamos de terminar a atribuição de responsabilidades a um ator?

Na figura 6.14 é apresenta a percentagem de atores sem elementos desconexos dentro da sua fronteira, de cada caso de estudo.

Para que a atribuição de responsabilidades a um ator possa ser considerada terminada, é necessário que existam elementos dentro da sua fronteira, e que esses elementos não estejam desconexos, ou seja, que apresentem algum tipo de ligação entre si. De notar que, para o cálculo da percentagem, o primeiro caso tem que se verificar. Assim sendo, os atores têm que ter pelo menos um elemento na sua fronteira.

A percentagem de atores sem elementos desconexos é bastante elevada. Com exceção dos sistemas HPA, NATS e BTW, o seu valor é de 100%, ou seja, todos os seus elementos possuem algum tipo de ligação.

Nenhum sistema é considerado extremo ou atípico, e o valor da mediana é elevado, o que sugere a atribuição de responsabilidades a um ator está terminada em praticamente todos os casos de estudo. No entanto, é aconselhável que os sistemas HPA,

NATS e BTW sejam analisados em mais detalhe, por forma a identificar elementos que se encontrem desconexos, e adicionar as respetivas ligações.

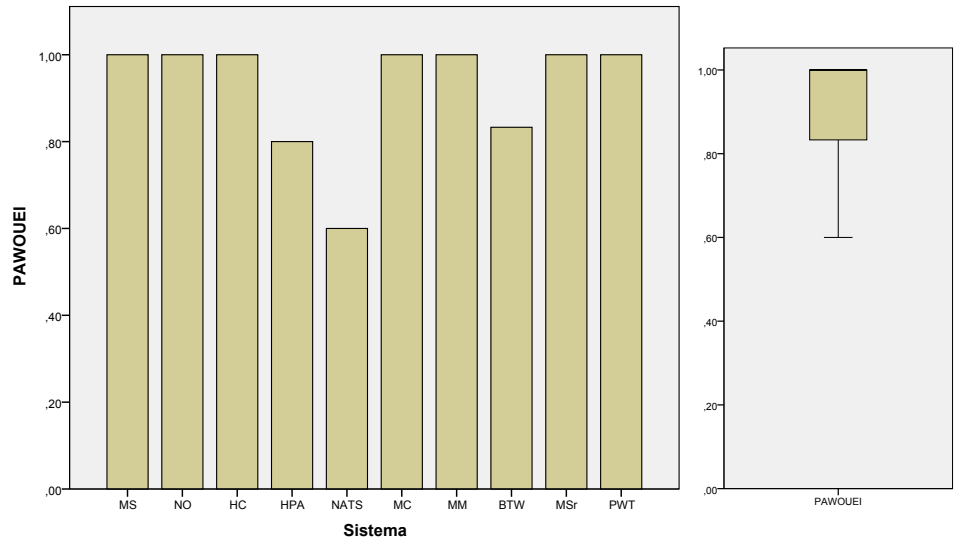


Figura 6.14: Percentagem de atores sem elementos desconexos dentro da sua fronteira

Questão 14 Quão perto estamos de terminar a atribuição de ligações aos atores?

Na figura 6.15 é apresentada a percentagem de atores com ligações de dependência e/ou de associação, de cada caso de estudo.

Para que a atribuição de ligações aos atores possa ser considerada terminada, é necessário que estes possuam pelo menos uma ligação de dependência ou uma ligação de associação. Atores sem nenhum destes tipos de ligação são considerados incompletos, uma vez que não é claro como é que estão conectados aos restantes atores do sistema.

A percentagem de atores com pelo menos uma ligação de dependência ou de associação é extremamente elevada. Com exceção dos sistemas HPA e NATS, o seu valor é de 100%, ou seja, todos os atores estão ligados a pelo menos um outro ator, quer seja através de ligações de dependências, quer seja por ligações de associações.

O valor da mediana é elevado, não existindo quartil inferior, quartil superior, ou bigodes. Apesar de a mediana se encontrar nos 100%, o que sugere que a atribuição de ligações aos atores está terminada em praticamente todos os casos de estudo, os sistemas HPA e NATS são considerados extremos. No entanto, o valor da sua percentagem também é elevado (perto de 90% em ambos os casos). Apesar disso, é aconselhável que estes sistemas sejam analisados em mais detalhe, por forma a identificar atores que se encontrem desconexos, e adicionar as respetivas ligações, tanto de dependência como de associação.

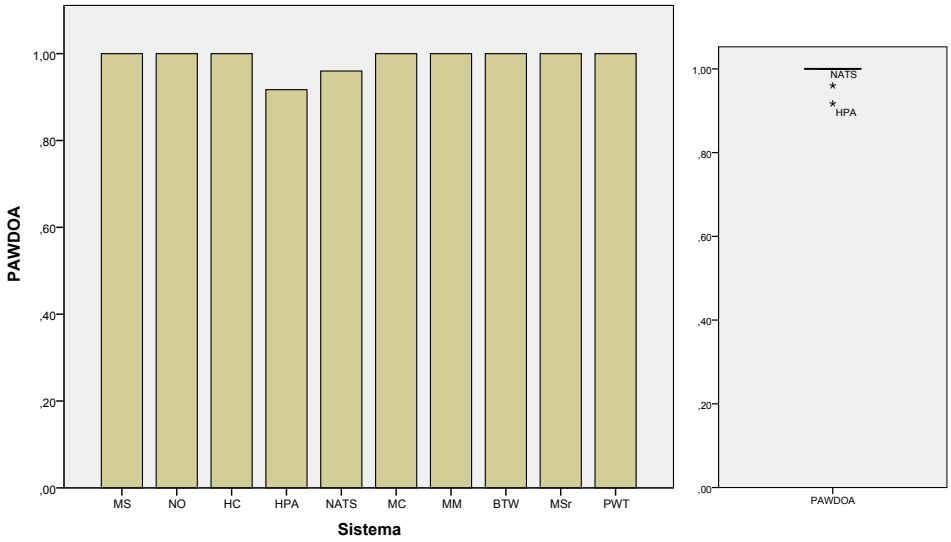


Figura 6.15: Percentagem de atores com ligações de dependência e/ou de associação

6.3.3 Resultados para o Objetivo Correção

Questão 15 Quão corretas são as associações?

Na figura 6.16 é apresentada a percentagem de atores com ligações de associação corretas, de cada caso de estudo.

A percentagem de ligações de associação corretas é de 100% em todos os casos de estudo. Quando o sistema não possui este tipo de ligações, o valor da percentagem é apresentado como 100%, pois a não existência de associações não faz com que estejam incorretas. Nos restantes casos, não existem ligações de associação incorretas.

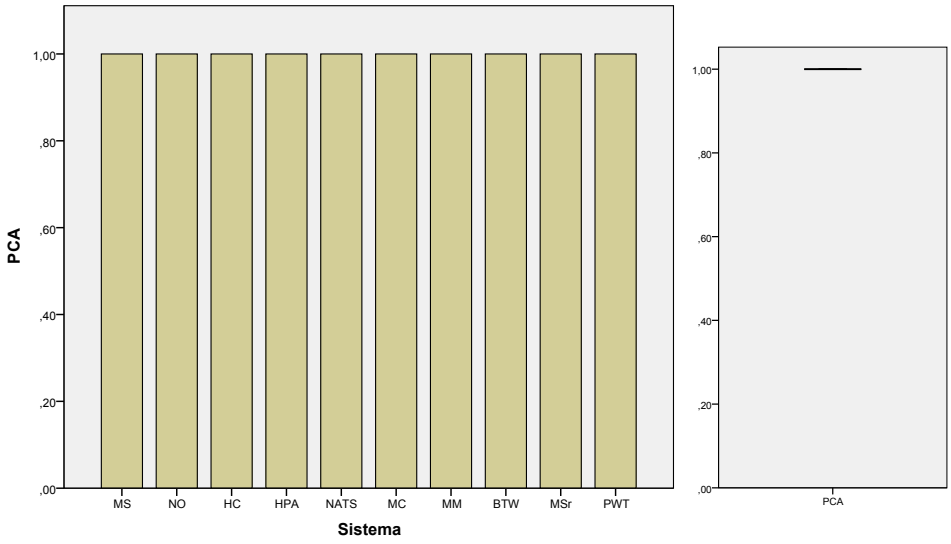


Figura 6.16: Percentagem de ligações de associação corretas

Questão 16 Quão corretas são as dependências?

Na figura 6.17 é apresentada a percentagem de atores com ligações de dependência corretas, de cada caso de estudo.

As dependências podem ser consideradas como as ligações mais importantes, tanto no modelo SD como no modelo SD, sendo fundamentais para a demonstração de ligações entre atores e captar relações estratégicas entre eles. A percentagem de ligações de dependência corretas é de 100% em todos os casos de estudo, pelo que não existem ligações incorretas em nenhum dos sistemas.

Torna-se importante referir que apenas no caso de estudo MS é que são apresentados tipos de ligações de dependência, nomeadamente o tipo ***critical***. Nos restantes casos de estudo, não são apresentados tipos de ligações de dependência, ou seja, todas as dependências são do tipo ***open***, não sendo utilizados os tipos ***committed*** e ***critical*** . Tal facto sugere um padrão de modelação, que pode indicar que o tipo de dependência não é uma preocupação na modelação dos diferentes sistemas.

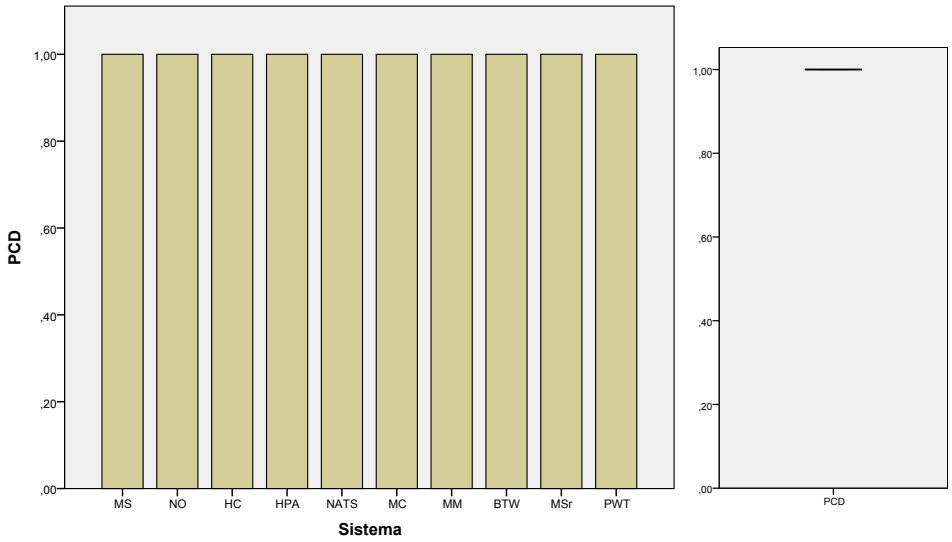


Figura 6.17: Percentagem de ligações de dependência corretas

Questão 17 Quão corretas são as contribuições?

Na figura 6.18 é apresentada a percentagem de ligações de contribuição corretas, de cada caso de estudo.

A percentagem de ligações de contribuição corretas é de 100%, em todos os casos de estudo, pelo que não existem ligações de contribuição incorretas em nenhum dos sistemas.

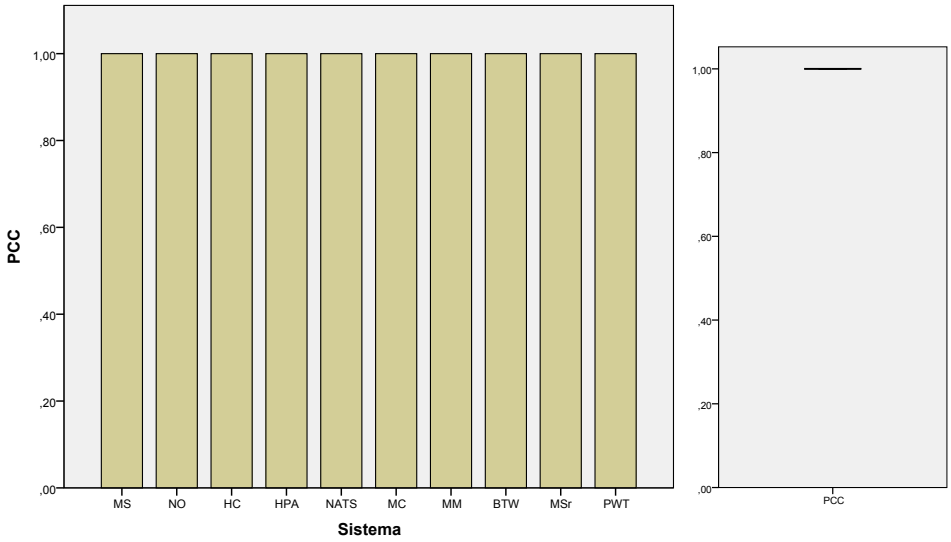


Figura 6.18: Percentagem de ligações de contribuição corretas

Questão 18 Quão corretas são as decomposições?

Na figura 6.19 é apresentada a percentagem de ligações de decomposição corretas, de cada caso de estudo.

A percentagem de ligações de decomposição corretas é extremamente elevada. Com exceção dos sistemas MS, HPA e NATS, o seu valor é de 100%, ou seja, todas as decomposições estão corretas. Apesar de a mediana ser elevada, o caso de estudo HPA é apresentado como atípico e o caso de estudo MS é apresentado como extremo. No entanto, o valor das percentagens também é elevado (acima de 90%).

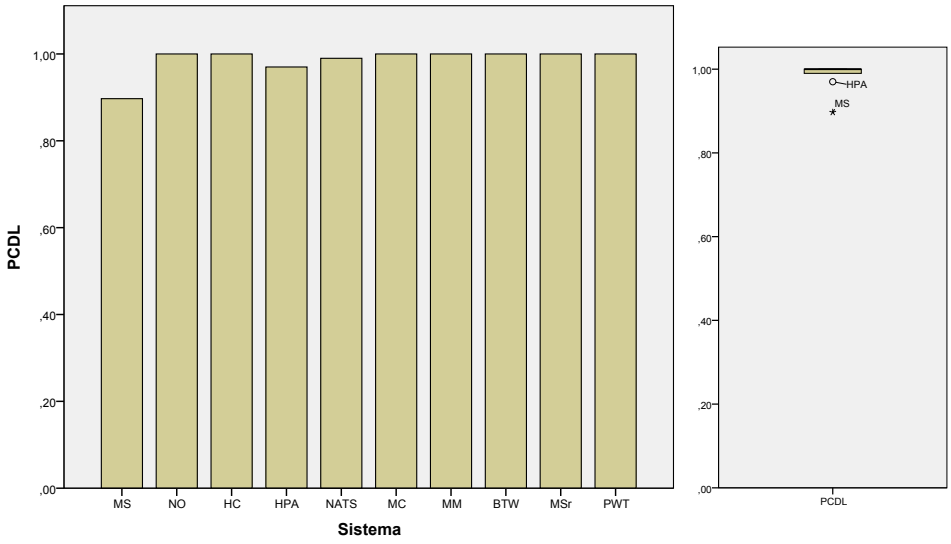


Figura 6.19: Percentagem de ligações de decomposição corretas

Questão 19 Quão correto é o posicionamento dos atores no modelo?

Na figura 6.20 é apresentada a percentagem de atores corretamente posicionados, de cada caso de estudo.

A percentagem de atores corretamente posicionados é elevada. Com exceção dos sistemas MS e NO, o seu valor é de 100%, ou seja, não existem atores dentro da fronteira de atores.

O valor da mediana é elevado, não existindo quartil inferior, quartil superior ou bigodes. Apesar de a mediana se encontrar nos 100%, o que sugere que o posicionamento dos atores é correto em praticamente todos os casos de estudo, os sistemas MS e NO são considerados extremos.

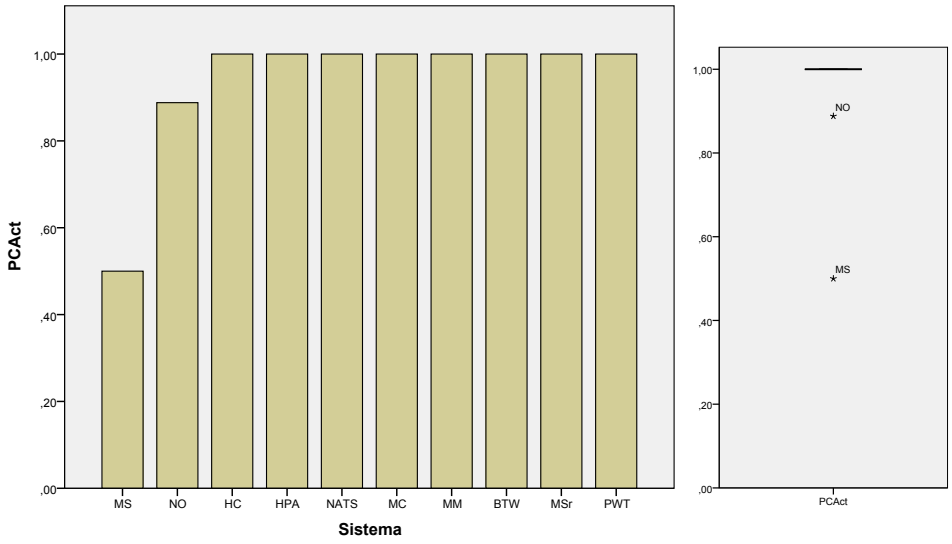


Figura 6.20: Percentagem de atores corretamente posicionados

6.4 Discussão

Para além da apresentação dos resultados obtidos, torna-se necessário realizar uma discussão sobre os mesmos. Essa discussão é feita para cada um dos objetivos (complexidade, completude e correção), tendo em conta os resultados obtidos em cada uma das questões.

6.4.1 Complexidade

Em relação ao objetivo complexidade, e tendo em conta os resultados obtidos para os sistemas analisados, observou-se que:

- O número de atores e de elementos varia de forma significativa nos diferentes sistemas. Tratando-se de sistemas com propósitos diversos, sendo que alguns são casos reais enquanto que outros são académicos, tal resultado era esperado.

- Existem valores extremos e atípicos no que se refere à responsabilidade de um ator num modelo, assim como nas decomposições associadas aos seus objetivos, *softgoals* e tarefas. Atores que apresentam esses valores devem ser analisados em mais detalhe, uma vez que podem estar a ocorrer 2 (duas) situações:

  1. **Ator demasiado complexo** – Candidato para uma futura decomposição em sub-atores, utilizando a associação *is-part-of*. No entanto, deve-se ter em atenção que, em alguns casos, a complexidade pode não ser acidental mas sim essencial. Nessas situações, esta análise continua a ter utilidade, uma vez que destaca atores essenciais que tenham associada uma complexidade essencial elevada. Tal facto pode indicar ao gestor de projeto que devem ser atribuídos mais recursos nas atividades de garantia de qualidade (inspeções e testes, por exemplo) em artefactos relacionados com a implementação de requisitos associados a este ator.
  2. **Excesso de decomposição por parte do engenheiro de *software*** – O engenheiro de *software* pode decompor demasiado os objetivos, *softgoals* e tarefas, por seguir uma estratégia de decomposição funcional. O nível de abstração do modelo diminui, e a inclusão de detalhes em demasia pode fazer com que o mesmo se torne mais difícil de compreender. Neste caso, abstrair decomposições detalhadas desnecessárias pode melhorar a compreensão geral do modelo de requisitos.

- O número de dependências de entrada e saída estão, de forma geral, distribuídas, ou seja, o seu número é semelhante nos sistemas como um todo. No entanto, a variação desse número é significativa nos diferentes atores de um sistema, não estando distribuída de forma uniforme. Casos em que um ator apenas possui dependências de entrada (poço), ou casos em que um ator apenas tem dependência de saída (fonte), ocorrem em quase todos os sistemas.

- Não foi possível obter valores em relação à variação na complexidade média dos diferentes tipos de atores. No entanto, não haver tipos específicos revela uma padrão de modelação que é discutido em mais detalhe na secção 6.4.2.

6.4.2 Completude

Em relação ao objetivo completude, e tendo em conta os resultados obtidos para os sistemas analisados, observou-se que:

- As especializações de atores não são utilizadas em nenhum dos sistemas analisados, apesar de tal ser considerado uma boa prática pela página *wiki* do *i**. Pode ser o caso de o engenheiro de *software* ter considerado que a utilização de atores especializados não iria acrescentar valor e informação útil ao modelo. No entanto, os

modelos devem ser analisados em mais detalhes, pois poderiam beneficiar da utilização de atores especializados, uma vez que tal faz aumentar a precisão e detalhe do modelo, tornando-o mais completo. Apesar disso, o facto de as especializações nunca serem utilizadas faz questionar a utilidade do mecanismo de especialização de atores.

- A maioria dos sistemas analisados apresenta uma percentagem elevada de objetivos detalhados, o que demonstra um bom nível de completude. Uma vez que um objetivo em si não fornece informação sobre como é que este é satisfeito, é aconselhável que todos os objetivos sejam decompostos através da utilização de ligações de *means-end*. Casos em que tal não aconteça devem ser analisados, devendo ser acrescentada(s) a(s) tarefa(s) que permita(m) satisfazer o objetivo.

- Grande parte dos sistemas analisados apresenta uma percentagem elevada de *softgoals* detalhados, o que demonstra um bom nível de completude. A decomposição e refinamento de *softgoals* são úteis para que estes se tornem mais concretos e para que decisões de desenho sejam incluídas no modelo, sendo aconselhável que todos os *softgoals* sejam refinados através da utilização de ligações de contribuição. Casos em que tal não aconteça devem ser analisados, devendo ser acrescentado(s) o(s) elemento(s) que permita(m) satisfazer o *softgoal*.

- Apesar de a diferença não ser muito significativa, verificou-se que, de forma geral, existe uma maior preocupação na operacionalização de tarefas do que na decomposição de objetivos. Tal pode dever-se ao facto de a operacionalização de tarefas fornecer informação mais detalhada sobre a implementação do sistema de *software*.

- No modelo SR, os atores apresentados no modelo SD devem ser "abertos", por forma a serem mostradas as suas intenções específicas e as suas responsabilidades. Assim sendo, os atores presentes no modelo SR devem ter elementos dentro da sua fronteira para serem considerados completos. Apenas 2 (dois) casos de estudo analisados apresentam elementos dentro da fronteira de todos os atores. Um ator vazio é um ator incompleto, pelo que é aconselhável que os restantes modelos sejam analisados de forma a compreender se os atores vazios são mesmo necessários. No entanto, é importante referir que os modelos são utilizados para comunicação e podem ter por objetivo o foco em determinada parte do sistema. Desta forma, alguns atores podem não ter elementos de maneira a diminuir a complexidade do modelo, e torná-lo mais fácil de compreender para o que se pretende comunicar.

- Por forma a se compreender como as dependências são satisfeitas entre atores, e como é que estes estão associados entre si, tanto o modelo SD como o modelo SR devem possuir pelo menos um destes tipos de ligações entre todos os seus atores. De forma geral, existe uma preocupação na atribuição de ligações aos atores nos sistemas analisados, o que permite aumentar o nível de completude do modelo como um todo.

6.4.3 Correção

Em relação ao objetivo correção, e tendo em conta os resultados obtidos para os sistemas analisados, observou-se que:

- Quando existem, as ligações de associação são modeladas de forma correta em todos os sistemas, o que sugere que este conceito é fácil de compreender e de modelar.
- As ligações de dependência são modeladas de forma correta em todos os sistemas, o que sugere que este conceito é fácil de compreender e de modelar. A não utilização de tipos de dependências em praticamente todos os casos de estudo revela um padrão de modelação. O tipo de dependência não é uma preocupação a nível de modelação, não sendo considerado se a não obtenção do *dependum* afeta, de alguma forma, o *depender* e o *dependee*. No entanto, também pode ser o caso em que o resultado da não obtenção do *dependum* não seja conhecido, ou que os engenheiros de *software* não compreendam como é que estes tipos podem ser utilizados. Verificando-se este último caso, pode existir um problema na especificação da linguagem *i** no que se refere a tipos de ligações de dependência.
- As ligações de contribuição são modeladas de forma correta em todos os sistemas, o que sugere que este conceito é fácil de compreender e de modelar.
- As ligações de decomposição são modeladas de forma correta em quase todos os sistemas. Nos casos em que existem incorreções, o seu número não é significativo, uma vez que a percentagem de correção se apresenta acima dos 90%, o que sugere que este conceito é relativamente fácil de compreender e de modelar.
- O posicionamento dos atores no modelo é modelado de forma correta em quase todos os sistemas. Nos casos em que existem incorreções, o seu valor é diferenciado. Num dos casos, o número de incorreções não é muito significativo, uma vez que a percentagem de correção se apresenta nos 90%. No entanto, no outro caso, a percentagem de incorreção corresponde a 50%, ou seja, metade dos atores encontram-se dentro da fronteira de outros atores. Apesar disso, é necessário ter em conta que estas incorreção resultam do facto de os sistemas em questão terem sido modelados utilizando o estilo *joint-venture* [Fux+01; KGM02], e não se devem a problemas de compreensão da linguagem.

6.4.4 Ameaças à Validade

Existem algumas ameaças e limitações à validade dos resultados das métricas que merecem ser consideradas e discutidas.

A primeira ameaça prende-se com o número de casos de estudo analisados. Devido ao facto de não existir um número a partir do qual se possa considerar suficiente, considera-se que quanto maior for o número de casos de estudo, mais precisos serão os

resultados obtidos. No entanto, tentou-se encontrar o máximo número de casos de estudo possível, e considera-se que 10 (dez) é um número razoável para se conseguir obter resultados válidos, uma vez que na área de engenharia de *software* os modelos completos não estão, normalmente, disponíveis para estudo.

Uma segunda ameaça tem que ver com o objetivo complexidade, e em que medida é que determinado modelo pode ser considerado complexo. Apesar das medições efetuadas, não existe um limite definido a partir do qual um modelo se pode considerar, de facto, complexo, e se essa complexidade afeta a sua compreensão. Por exemplo, não existe um limite a partir do qual as decomposições de tarefas se tornam complexas de forma a que tal dificulte a boa compreensão do modelo. Idealmente, teria que se identificar o valor desses limites, e realizar experiências com engenheiros de *software*, por forma a determinar até que ponto é que os limites identificados estão relacionados com uma maior dificuldade em compreender totalmente o modelo *i**.

Outra ameaça está relacionada com a própria amostra de casos de estudo. Os casos de estudo utilizados são, no geral, considerados como bons exemplos de modelos de requisitos *i**, tendo vários deles sido revistos por pares, eliminando eventuais problemas detetados na sua forma original. Assim sendo, podem ser utilizados como referência de boas práticas de modelação. Para a métrica de correção, seria importante a existência de maus exemplos, ou seja, casos de estudo em que o número de incorreções fosse mais elevado. Dessa forma, seria mais facilmente detetável a existência de problemas na compreensão da linguagem *i**, assim como determinados padrões de modelação. A dificuldade em encontrar esses maus exemplos resulta da não disponibilização pública de modelos de requisitos, por parte dos seus autores.

Por fim, a existência de diferentes dialetos e a não conformidades na utilização da linguagem, ou seja, as variações do *framework i**, fazem com que se torne mais difícil o processo de avaliação da complexidade, completude e correção. Adicionalmente, para além dos dialetos formais, existem dialetos informais que resultam das práticas da organização onde o caso de estudo foi desenvolvido. Apesar de tal facto dificultar a avaliação dos atributos de qualidade, em última instância os modelos são utilizados para comunicação, pelo que o seguimento de todas as regras do *framework i** pode não ser um requisito fundamental, na medida em que a compreensão por parte dos *stakeholders* é o mais importante, independentemente da forma como os diferentes elementos são modelados.

Apesar da existência das ameaças e limitações apresentadas, para fins de avaliação e de validação do conjunto de métricas proposto, os casos de estudo propostos abrangem as situações que se pretendem tratar.

6.5 Validação Teórica das Métricas da Complexidade

O conjunto de trabalhos em métricas de *software* atualmente existente, inclui abordagens para a sua validação, quer teórica quer empírica. Nas secções anteriores, foi tratada a validação empírica das métricas propostas. Nesta secção, é discutida a validação

teórica das métricas da complexidade, recorrendo ao conjunto de propriedades proposto por Weyuker em [Wey88]. Note-se que este conjunto de propriedades foi especificamente proposto para endereçar a complexidade, não sendo adequado para a avaliação de outros atributos de qualidade, como completude ou correção. Deste modo, a validação teórica das métricas propostas nesta dissertação, é feita apenas para as métricas relativas à complexidade.

Weyuker propõe um conjunto de 9 (nove) propriedades para as métricas de complexidade de um programa de *software*. Essas propriedades foram frequentemente adotadas para validar não só as métricas de programas de *software*, como também as de documentos de desenho, tal como diagramas de classes. Para uma correta validação das métricas propostas, tornou-se necessário adaptar essas propriedades ao contexto de métricas de complexidade para modelos GORE.

A tabela 6.1 apresenta as propriedades de Weyuker, adaptadas para as métricas de complexidade definidas nesta dissertação. As propriedades são apresentadas tanto em linguagem natural como formalmente. Na tabela, considere-se P, Q e R como modelos. Sejam $|P|$, $|Q|$ e $|R|$ as suas complexidades, respetivamente, e seja $|P;Q|$ a complexidade resultante de $|P|$ composto com $|Q|$.

Tabela 6.1: Propriedades de Weyuker, adaptadas de [Wey88]

#	Propriedades de Weyuker
1	Existem modelos diferentes cujo valor da complexidade é diferente para a mesma métrica: $\exists P, \exists Q : P \neq Q \wedge \|P\| \neq \|Q\|$
2	Existe apenas um número finito de modelos com complexidade c. Seja c um número não negativo, S o conjunto de modelos com complexidade c, e n o numeral cardinal do conjunto: S $\forall c \in \mathbb{R}_0^+, \forall P : \|P\| = c \Rightarrow P \in S, \exists n \in \mathbb{N}_0 : \#S = n$
3	Modelos diferentes podem ter a mesma complexidade: $\exists P, \exists Q : P \neq Q \wedge \|P\| = \|Q\|$
4	Modelos diferentes que sejam funcionalmente equivalentes pode ter complexidades diferentes: $\exists P, \exists Q : P \equiv Q \wedge \|P\| \neq \|Q\|$
5	Monotonia é uma propriedade fundamental de todas as medidas de complexidade. Um modelo em isolamento é, no máximo, tão complexo quanto a sua composição com outro modelo: $\forall P, \forall Q : \|P\| \leq \|P;Q\| \wedge \|Q\| \leq \|P;Q\|$
6	As complexidades resultantes de compor o mesmo modelo (R) com dois modelos diferentes com a mesma complexidade (P e Q), não são necessariamente iguais. Inversamente, as complexidades de compor dois modelos diferentes (P e Q) com a mesma complexidade com um terceiro modelo (R), não são necessariamente iguais: $\exists P, \exists Q, \exists R : P \neq Q \wedge \|P\| = \|Q\| \wedge \|P;R\| \neq \|Q;R\|$ $\exists P, \exists Q, \exists R : P \neq Q \wedge \|P\| = \|Q\| \wedge \|R;P\| \neq \|R;Q\|$
7	A complexidade no modelo deve ser sensível à organização dos seus elementos. Sejam P e Q dois modelos tais que Q seja formado pela permutação da ordem dos elementos em P. Seja $Perm()$ a operação de permutação: $\exists P, \exists Q : Q = Perm(P) \wedge \|P\| \neq \|Q\|$
8	Se um modelo é uma mudança de nome de outro modelo, então a sua complexidade deve ser a mesma. Seja $Rename()$ a operação que transforma o modelo P na sua versão renomeada: Q $\exists P, \exists Q : Q = Rename(P) \wedge \|P\| = \|Q\|$
9	A complexidade da composição de dois modelos pode ser maior do que a soma das complexidades desses modelos. A complexidade extra pode resultar da interação entre os modelos: $\exists P, \exists Q : \|P\| + \|Q\| < \|P;Q\|$

A tabela 6.2 apresenta a aplicação das propriedades de Weyuker às métricas da complexidade. Cada métrica foi aplicada a cada uma das 9 (nove) propriedades apresentadas anteriormente, sendo que *"Sim"* significa que a métrica obedece à propriedade, e *"Não"* significa que não obedece. De notar que, devido ao facto de as métricas correspondentes à questão Q8 (`NEIAgentB`, `NEIPosB` e `NEIRoleB`) não poderem ser aplicadas, uma vez que não existem atores com tipos específicos em nenhum dos modelos, essas métricas não foram consideradas na validação teórica.

Tabela 6.2: Aplicação das propriedades de Weyuker às métricas da complexidade

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
NAct	Sim	Sim	Sim	Sim	Sim	Sim	**Não**	Sim	**Não**
NElem	Sim	Sim	Sim	Sim	Sim	Sim	**Não**	Sim	**Não**
NEIAB	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	**Não**
MinNEIAB	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	**Não**
MaxNEIAB	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	**Não**
AvgNEIAB	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	**Não**
NDAG	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	**Não**
MinNDAG	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	**Não**
MaxNDAG	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	**Não**
AvgNDAG	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	**Não**
NDAS	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	**Não**
MinNDAS	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	**Não**
MaxNDAS	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	**Não**
AvgNDAS	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	**Não**
NDAT	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	**Não**
MinNDAT	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	**Não**
MaxNDAT	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	**Não**
AvgNDAT	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	**Não**
PODA	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	**Não**
PIDA	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	**Não**

A maioria das métricas propostas obedece às propriedades de Weyuker, existindo algumas exceções. Esta forte adesão às propriedades de Weyuker resulta de o conjunto de métricas proposto ser de natureza estrutural, baseado em contagens de elementos: a adição ou remoção de elementos nos modelos tem um impacto direto na maioria das propriedades, alinhado com o definido nas propriedades de Weyuker. A propriedade 9 apresenta a maior exceção, sendo que nenhuma das métricas obedece a esta propriedade. Assumiu-se que a composição de 2 (dois) modelos não introduz elementos que não estejam presentes em pelo menos um desses modelos. Assim, modelos compostos não podem apresentar um valor superior ao da soma das suas partes. Sem este pressuposto, a propriedade 9 seria verificável para todas as métricas. As métricas `NAct` e `NElem` não obedecem à propriedade 7. Tal deve-se ao facto de que uma reorganização no modelo que não introduza novos atores nem elementos, não afeta o seu número, logo, não altera a sua complexidade. Tendo em conta que se encontraram casos em que as métricas não

obedecem às propriedades, torna-se necessário referir que é comum encontrar métricas de complexidade que não obedeçam a todas as propriedades de Weyuker [MA08], em especial no que se refere à propriedade 9 [GR01; KB07; MR13].

6.6 Sumário

Neste capítulo foram analisados diversos casos de estudo, os quais foram modelados com a ferramenta desenvolvida para a presente dissertação.

Os casos de estudo começaram por ser apresentados, tendo sido fornecida informação sobre a sua adaptação à normal do *framework i**. Os resultados obtidos pela aplicação das métricas aos casos de estudo foram analisados e discutidos, tendo sido feitas algumas observações e retiradas certas conclusões referentes a cada questão em particular, e a cada objetivo em geral. Foram, também, identificadas ameaças e limitações à validade dos resultados obtidos.

Por fim, foi feita uma avaliação teórica das métricas referentes à complexidade, através de propriedades de Weyuker.

7

Conclusões

Este capítulo começa por apresentar informação sobre as principais contribuições da presente dissertação e a sua importância na área de engenharia de requisitos. Depois, são apresentas as suas limitações. Por fim, é descrito o trabalho futuro.

7.1 Contribuições

As principais contribuições desta dissertação prendem-se com a definição, recolha e avaliação de métricas para modelos de requisitos, de forma a dar suporte à avaliação quantitativa de modelos de requisitos *i**. Seguindo a abordagem GQM, foi proposto um conjunto de métricas para avaliar a complexidade, completude e correção de modelos *i**. As métricas foram descritas tanto informalmente, em linguagem natural, como formalmente, através da linguagem OCL. Uma versão preliminar do conjunto de métricas descrito nesta dissertação foi aceite na conferência CAiSE'14 [GGA14], onde ganhou o prémio de melhor artigo (*Best Paper Award*). As restantes métricas serão também alvo de publicação, estando em fase de revisão uma versão aumentada do artigo aceite na conferência CAiSE'14, submetida ao *journal* Information Systems da Elsevier. Este último artigo resultada de um convite feito pelos organizadores do CAiSE'14, sendo que o *journal* apresenta o título *Special Issue: Advanced Information Systems Engineering (CAiSE'14)*.

As métricas foram incorporadas numa ferramenta de modelação, de maneira a serem recolhidas de forma automática. A ferramenta foi implementada de raiz para a presente dissertação, tendo sido utilizados os *plugins* EMG e GMF do IDE Eclipse. Estes *plugins* permitem a integração de restrições OCL ao meta-modelo da ferramenta, assim como uma apresentação textual e gráfica dos modelos *i**. Antes da criação da ferramenta, foi

realizado um estudo das ferramentas presentes na página *wiki* do *i**, por forma a identificar aspetos positivos, aspetos que podem ser melhorados, e aspetos que não estão a ser tido em conta. Esse estudo foi aceite no *International i* Workshop* 2013, co-localizado com a conferência CAiSE'13 [GGA13].

Por fim, as métricas propostas passaram por uma avaliação experimental e por uma a validação teórica. Para a avaliação experimental, as métricas foram aplicadas a um conjunto de casos de estudo reais e académicos. Para a validação teórica, foram aplicadas, às métricas da complexidade, as propriedades de Weyuker. A avaliação da complexidade, completude e correção pode contribuir para uma melhor compreensão dos modelos de requisitos *i**, podendo ser utilizada por forma a melhorar a qualidade global dos mesmos. Através da obtenção de medidas relativas à qualidade, é possível que sejam estabelecidos objetivos de melhoria no próprio processo de desenvolvimento de um sistema de *software*, por forma a se conseguir reduzir ou eliminar causa de problemas que afetam esse sistema.

7.2 Limitações

Apesar de se cobrir de forma relativamente extensa a complexidade, a completude e a correção de modelos *i**, existem alguns aspetos que apresentam algumas limitações.

A principal limitação prende-se com o facto de as métricas serem aplicadas ao modelo como um todo, não existindo diferença entre modelo SD e modelo SR. A separação clara destes dois tipos de modelos iria permitir que as métricas da complexidade, por exemplo, fossem mais completas, havendo métricas específicas para cada um e métricas comuns aos dois modelos.

A instalação do protótipo é relativamente complexa, envolvendo a utilização de diversos *plugins* externos, sendo aconselhável que se siga o manual descrito no anexo B. Sendo o propósito do protótipo fazer uma demonstração de conceito, há aspetos de usabilidade que requerem algum trabalho adicional. Um aspeto é a configuração, na própria ferramenta, de quais as métricas a serem apresentadas ao utilizador. Outro aspeto prende-se com a utilização de um processo de instalação simplificado para utilizadores menos experientes em ambientes Eclipse. O próprio nível de usabilidade da ferramenta não é analisado, pelo que não se sabe até que ponto a ferramenta é fácil de utilizar e quais as principais dificuldades por parte do utilizador. Por fim, a ferramenta não permite a distinção entre modelo SD e modelo SR e não existem vistas parciais, que seriam úteis no caso de se querer analisar apenas partes específicas do modelo.

Uma última limitação relativamente à ferramenta, prende-se com o facto de ser apresentado um valor para cada uma das métricas, sendo que a sua análise tem que ser feita com auxílio de um programa externo. Desta forma, existe a necessidade de extrair as métricas para se proceder a uma análise mais aprofundada das mesmas.

7.3 Trabalho Futuro

Para trabalho futuro, pretende-se encontrar formas de identificar oportunidades de melhoria dos modelos de requisitos *i**, com base nas métricas propostas nesta dissertação, através da integração das métricas com catálogos de padrões e soluções de refabricação.

Também se pretende estender o conjunto de métricas proposto para cobrir outros atributos de qualidade, assim como realizar a avaliação para um maior número de modelos de requisitos *i**. Torna-se interessante a análise de bons e maus exemplos de modelos, assim como uma comparação dos resultados para cada um deles.

A nível da ferramenta, um aspeto a ter em conta é a implementação de vistas, sendo possível separar modelos SD e modelos SR, analisando cada um deles em separado, e havendo uma seleção de métricas a serem apresentadas conforme se trate de um ou de outro modelo. As vistas parciais também são úteis dentro dos próprios modelos SD e SR, no caso de se querer analisar apenas partes específicas do modelo. Neste ponto, também é importante a possibilidade do utilizador escolher que métricas pretende que sejam apresentadas. Adicionalmente, também se torna útil que quando o utilizador seleciona um elemento, apareçam apenas as métricas relevantes para esse mesmo elemento.

Ainda relativamente à ferramenta, um aspeto a ter em conta prende-se com a análise das métricas apresentadas. Embora as métricas possam ser extraídas e analisadas com auxílio de um programa externo, e de a extração de métricas ser feita de forma automática, este processo pode ser melhorado. A apresentação de vistas tipo *boxplot* dentro da própria ferramenta, para dar contexto às métricas, ajudaria os engenheiros de *software* a avaliar potenciais problemas detetados com as métricas, sem terem que recorrer a uma ferramenta externa de estatística.

Um outro aspeto a nível da ferramenta, é a existência de exemplos por forma a facilitar a aprendizagem da linguagem *i**. De maneira a se compreender que exemplos é que podem ajudar na aprendizagem, também se torna interessante a realização de um estudo sobre a evolução e a facilidade de aprendizagem do *framework i**. Para tal, pretende-se analisar trabalhos de alunos de disciplinas universitárias onde o *framework i** seja lecionado. Idealmente, os trabalhos seriam divididos em fases, e os modelos criados nas diferentes fases seriam analisados, sendo feita uma avaliação da sua evolução, tanto a nível de complexidade, como de completude e correção. Com dados concretos sobre quais os erros mais comuns quando se está a aprender, mais fácil se torna a criação de exemplos que ajudem na aprendizagem. Para além disso, a identificação de dificuldades comuns podem revelar aspetos da linguagem que não são intuitivos.

Por forma a que a ferramenta se torne fácil e intuitiva de instalar, pretende-se realizar a conversão da mesma através de Eclipse RCP (do inglês *Rich Client Platform*) [Vog13; Ecl14l]. O Eclipse RCP permite que o projeto da ferramenta seja convertido num programa independente, sem a necessidade de importação para o IDE Eclipse, instalando-se como um programa normal no computador.

Por fim, pretende-se identificar limites a partir dos quais um elemento seja considerado demasiado complexo, de forma a que essa complexidade afete a sua compreensão. Para além da identificação do valor desses limites, pretende-se realizar experiências com engenheiros de *software*, por forma a determinar até que ponto é que os limites identificados estão relacionados com uma maior dificuldade em compreender totalmente o modelo *i**.

Bibliografia

[Abr+04] A. Abran, J. W. Moore, P. Bourque, R. Dupuis e L. L. Tripp. *Guide to the Software Engineering Body of Knowledge (SWEBOK)*. Los Alamitos, EUA: IEEE Computer Society, 2004. ISBN: 0-7695-2330-7.

[Abr01] F. B. e Abreu. *Using OCL to Formalize Object Oriented Metrics Definitions*. Rel. téc. ES007/2001. Universidade Nova de Lisboa e Instituto de Engenharia de Sistemas e Computadores, 2001.

[Agi14] Agile Methodology. *What is the Agile Software Development Methodology?* Último acesso em Setembro de 2014. URL: http://agilemethodology.org/.

[Ale+06] F. Alencar, A. Moreira, J. Araújo, J. Castro, C. Silva e J. Mylopoulos. "Using Aspects to Simplify i* Models". Em: *Proceedings of the 14th IEEE International Requirements Engineering Conference*. IEEE Computer Society, 2006.

[Ale+08] F. Alencar, C. Silva, M. Lucena, J. C. E. Santos e R. Ramos. "Improving the Understandability of i* Models". Em: *International Conference on Enterprise Information Systems*. Vol. 3. 1. 2008, pp. 129–136. ISBN: 978-989-8111-38-8.

[Amy+10] D. Amyot, S. Ghanavati, J. Horkoff, G. Mussbacher, L. Peyton e E. Yu. "Evaluating goal models within the goal-oriented requirement language". Em: *International Journal of Intelligent Systems* 25.8 (2010), pp. 841–877.

[An+09] Y. An, P. W. Dalrymple, M. Rogers, P. Gerrity, J. Horkoff e E. Yu. "Collaborative Social Modeling for Designing a Patient Wellness Tracking System in a Nurse-managed Health Care Center". Em: *Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology*. DESRIST '09. Philadelphia, Pennsylvania: Association for Computing Machinery, 2009, 2:1–2:14. ISBN: 978-1-60558-408-9.

[Ara+05] J. Araújo, E. Baniassad, P. Clements, A. Moreira, A. Rashid e B. Tekinerdoğa. *Early Aspects: The Current Landscape*. Rel. téc. Lancaster University, 2005.

[Aya+05] C. P. Ayala, C. Cares, J. P. Carvallo, G. Grau, M. Haya, G. Salazar, X. Franch, E. Mayol e C. Quer. *A Comparative Analysis of i*-Based Agent-Oriented Modeling Language*. 2005.

[BD09] A. T. Bahill e F. F. Dean. "Discovering System Requirements". Em: *Handbook of Systems Engineering and Management*. Ed. por A. P. Sage e W. B. Rouse. 2ª ed. John Wiley & Sons, Inc., 2009.

[BBA02] A. L. Baroni, S. Braz e F. B. e Abreu. "Using OCL to Formalize Object-Oriented Design Metrics Definitions". Em: *Proceedings of the International Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE'02) at ECOOP'02*. ECOOP'02. Springer-Verlag, 2002.

[BCR94] V. R. Basili, G. Caldiera e H. D. Rombach. *The Goal Question Metric Paradigm, Encyclopedia of Software Engineering*. 1ª ed. Hoboken, EUA: John Wiley & Sons, Inc., 1994. ISBN: 1-54004-8.

[BCR14] V. R. Basili, G. Caldiera e H. D. Rombach. *The Goal Question Metric Approach*. Acedido em Setembro de 2014. URL: ftp://ftp.cs.umd.edu/pub/sel/papers/gqm.pdf.

[Bec99] K. Beck. "Embracing Change with Extreme Programming". Em: *Computer* 32.10 (1999), pp. 70–77.

[Boe86] B. W. Boehm. "A Spiral Model of Software Development and Enhancement". Em: *ACM SIGSOFT Software Engineering Notes* 11.4 (1986), pp. 14–24.

[Boo+07] G. Booch, R. A. Maksimchuk, M. W. Engle, B. J. Young, J. Conallen e K. A. Houston. *Object-Oriented Analysis and Design with Applications*. 3ª ed. EUA: Addison-Wesley Professional, 2007. ISBN: 978-0201895513.

[Bor09] C. C. Borba. "Uma Abordagem Orientada a Objetivos para as Fases de Requisitos de Linhas de Produtos de Software". Tese de mestrado. Brasil: Centro de Informática, Universidade Federal de Pernambuco, 2009.

[BHX09] C. C. Borba, J. Henrique e L. Xavier. *BTW – If You Go, My Advice to You*. Centro de Informática, Universidade Federal de Pernambuco, Brasil. SCORE Contest, 2009.

[Bro87] F. P. Brooks. "No Silver Bullet – Essence and Accidents of Software Engineering". Em: *Computer* 20.4 (1987), pp. 10–19. ISSN: 0018-9162.

[CKM01] J. Castro, M. Kolp e J. Mylopoulos. "A Requirements-Driven Development Methodology". Em: *Proceedings of the 13th International Conference on Advanced Information Systems Engineering*. CAiSE '01. Londres, Reino Unido: Springer-Verlag, 2001, pp. 108–123. ISBN: 3-540-42215-3.

[Cen14] Center for Disease Control and Prevention. *Guidelines for Evaluating Surveillance Systems*. Último acesso em Setembro de 2014. URL: http://www.cdc.gov/mmwr/preview/mmwrhtml/00001769.htm.

[CG14] Center for Disease Control and Prevention e Guidelines Working Group. *Updated Guidelines for Evaluating Public Health Surveillance Systemss*. Último acesso em Setembro de 2014. URL: http://www.cdc.gov/mmwr/preview/mmwrhtml/rr5013a1.htm.

[CMM14] CMMI Institute. *CMMi – Capability Maturaty Model Integration*. Último acesso em Setembro de 2014. URL: http://cmmiinstitute.com/.

[CF14] D. Colomer e X. Franch. "StarGro: Building i* Metrics for Agile Methodologies". Em: *7th International i* Workshop (iStar 2014)*. Ed. por F. Dalpiaz e J. Horkoff. Vol. 1157. CEUR Workshop Proceedings. Thessaloniki, Greece, 2014.

[DLF93] A. Dardenne, A. van Lamsweerde e S. Fickas. "Goal-Directed Requirements Acquisition". Em: *Science of Computer Programming* 20.1-2 (1993), pp. 3–50.

[Des14] DesCARTES. *DesCARTES Architect*. Último acesso em Setembro de 2014. URL: http://www.isys.ucl.ac.be/descartes/.

[Dia09] A. Dias. "Uma Linguagem Específica do Domínio para uma Abordagem Orientada aos Objetivos Baseada em KAOS". Tese de mestrado. Portugal: Departamento de Informática, Universidade Nova de Lisboa, 2009.

[Ecl14a] Eclipse. *Eclipse Graphical Modeling Framework (GMF) Tooling*. Último acesso em Setembro de 2014. URL: http://eclipse.org/gmf-tooling/.

[Ecl14b] Eclipse. *Eclipse Modeling Framework Project (EMF)*. Último acesso em Setembro de 2014. URL: http://eclipse.org/modeling/emf/.

[Ecl14c] Eclipse. *Eclipse Modeling Tools*. Último acesso em Setembro de 2014. URL: http://www.eclipse.org/downloads/packages/eclipse-modeling-tools/keplersr1.

[Ecl14d] Eclipse. *Emfatic*. Último acesso em Setembro de 2014. URL: http://wiki.eclipse.org/Emfatic.

[Ecl14e] Eclipse. *Epsilon*. Último acesso em Setembro de 2014. URL: https://www.eclipse.org/epsilon/.

[Ecl14f] Eclipse. *Epsilon Generation Language*. Último acesso em Setembro de 2014. URL: http://www.eclipse.org/epsilon/doc/egl/.

[Ecl14g] Eclipse. *Epsilon Object Language*. Último acesso em Setembro de 2014. URL: http://www.eclipse.org/epsilon/doc/eol/.

[Ecl14h] Eclipse. *Epsilon Validation Language*. Último acesso em Setembro de 2014. URL: http://www.eclipse.org/epsilon/doc/evl/.

[Ecl14i] Eclipse. *EuGENia*. Último acesso em Setembro de 2014. URL: http://www.eclipse.org/epsilon/doc/eugenia/.

[Ecl14j] Eclipse. *IDE Eclipse*. Último acesso em Setembro de 2014. URL: https://www.eclipse.org/ide/.

[Ecl14k] Eclipse. *OCL/OCLinEcore*. Último acesso em Setembro de 2014. URL: http://wiki.eclipse.org/OCL/OCLinEcore.

[Ecl14l] Eclipse. *Rich Client Platform*. Último acesso em Setembro de 2014. URL: http://wiki.eclipse.org/index.php/Rich_Client_Platform.

[Eng09] J. Engmann. "Evaluating the Impact of Evolving Requirements on Wider System Goals: Using i* Methodology Integrated with Satisfaction Arguments to Evaluate the Impact of Changing Requirements in HIV/AIDS Monitoring Systems in the UK". Tese de mestrado. Inglaterra: School of Informatics, City University of London, 2009.

[Esp12] P. Espada. "Melhoria da qualidade para modelos orientados a objectivos: o caso da abordagem KAOS". Tese de mestrado. Portugal: Departamento de Informática, Universidade Nova de Lisboa, 2012.

[EGA11] P. Espada, M. Goulão e J. Araújo. "Measuring complexity and completeness of KAOS goal models". Em: *Proceedings of the International Workshop on Empirical Requirements Engineering (EmpiRE 2011), at the 19th International Requirements Engineering Conference (RE 2011)*. IEEE Computer Society, 2011, pp. 29–32.

[EGA13] P. Espada, M. Goulão e J. Araújo. "A Framework to Evaluate Complexity and Completeness of KAOS Goal Model". Em: *Proceedings of the 25th International Conference on Advanced Information Systems Engineering (CAiSE'13)*. Vol. 7908. Lecture Notes in Computer Science. Springer-Verlag, 2013, pp. 562–577. ISBN: 978-3-642-38708-1.

[Esp+09] S. España, N. Condori-Fernandez, A. Gonzalez e Ó. Pastor. "Evaluating the Completeness and Granularity of Functional Requirements Specifications: A Controlled Experiment". Em: *17th International Requirements Engineering Conference*. Atlanta, Georgia, EUA: IEEE Computer Society, 2009, pp. 161–170.

[Fig+08] E. Figueiredo, N. Cacho, C. Sant'Anna, M. Monteiro, U. Kulesza, A. Garcia, S. Soares, F. Ferrari, S. Khan, F. C. Filho e F. Dantas. "Evolving Software Product Lines with Aspects: An Empirical Study on Design Stability". Em: *Proceedings of the 30th International Conference on Software Engineering*. ICSE '08. Leipzig, Alemanha: Association for Computing Machinery, 2008, pp. 261–270. ISBN: 978-1-60558-079-1.

[Fow11] M. Fowler. *Domain-Specific Languages*. USA: Addison-Wesley Professional, 2011. ISBN: 0-321-71294-3.

[Fra08] X. Franch. "A Metrics Definition Framework for i*". Em: *Proceedings of the 3rd International i* Workshop*. 2008, pp. 25–28.

[Fra09] X. Franch. "A Method for the Definition of Metrics over i* Models". Em: *Proceedings of the 21st International Conference on Advanced Information Systems Engineering*. CAiSE '09. Amesterdão, Holanda: Springer-Verlag, 2009, pp. 201–215. ISBN: 978-3-642-02143-5.

[FG08] X. Franch e G. Grau. "Towards a Catalogue of Patterns for Defining Metrics over i* Models". Em: *Proceedings of the 20th International Conference on Advanced Information Systems Engineering*. CAiSE '08. Berlim, Heidelberg: Springer-Verlag, 2008, pp. 197–212. ISBN: 978-3-540-69533-2.

[Fux+01] A. Fuxman, P. Giorgini, M. Kolp e J. Mylopoulos. "Information Systems as Social Structures". Em: *Proceedings of the International Conference on Formal Ontology in Information Systems - Volume 2001*. FOIS '01. Ogunquit, Maine, EUA: Association for Computing Machinery, 2001, pp. 10–2. ISBN: 1-58113-377-4.

[Gal04] D. Galin. *Software Quality Assurance. From theory to implementation*. 1ª ed. EUA: Addison-Wesley, 2004. ISBN: 0-201-70945-7.

[Gia+10] G. Giachetti, F. Alencar, X. Franch e Ó. Pastor. *Applying i* Metrics for the Integration of Goal-Oriented Modeling into MDD Processes*. Rel. téc. Universitat Politècnica de Catalunya, Espanha, 2010.

[GMS05] P. Giorgini, J. Mylopoulos e R. Sebastiani. "Goal-oriented Requirements Analysis and Reasoning in the Tropos Methodology". Em: *Engineering Applications of Artificial Intelligence* 18.2 (2005), pp. 159–171. ISSN: 0952-1976.

[Gou08] M. Goulão. "Component-Based Software Engineering: a Quantitative Approach". Tese de doutoramento. Portugal: Departamento de Informática, Universidade Nova de Lisboa, 2008.

[GGA13] C. Gralha, M. Goulão e J. Araújo. "A Systematic Comparison of i* Modelling Tools Based on Syntactic and Well-formedness Rules". Em: *Proceedings of the 6th International i* Workshop 2013*. Vol. 978. CEUR Workshop Proceedings. Valência, Espanha: CEUR-WS.org, 2013, pp. 43–48.

[GGA14] C. Gralha, M. Goulão e J. Araújo. "Identifying Modularity Improvement Opportunities in Goal-Oriented Requirements Models (Best Paper Award)". Em: *26th International Conference on Advanced Information Systems Engineering (CAiSE'14)*. Salónica, Grécia: Springer, 2014, pp. 91–104.

[GR01] Gursaran e G. Roy. "On the Applicability of Weyuker Property 9 to Object-Oriented Structural Inheritance Complexity Metrics". Em: *IEEE Transactions on Software Engineering* 27.4 (2001), pp. 381–384.

[Hea14] Health Protection Agency. *HPA - Public Health England's National Health Protection Service*. Último acesso em Setembro de 2014. URL: http://www.hpa.org.uk/.

[HY12a] A. Hilts e E. Yu. "Design and Evaluation of the Goal-oriented Design Knowledge Library Framework". Em: *Proceedings of the 2012 iConference*. iConference '12. Toronto, Ontário, Canadá: Association for Computing Machinery, 2012, pp. 384–391. ISBN: 978-1-4503-0782-6.

[HY09] J. Horkoff e E. Yu. "Evaluating Goal Achievement in Enterprise Modeling - An Interactive Procedure and Experiences". Em: *Proceedings of The Practice of Enterprise Modeling*. Vol. 39. Lecture Notes in Business Information Processing. 2009, pp. 145–160. ISBN: 978-3-642-05351-1.

[HY12b] J. Horkoff e E. Yu. "Comparison and Evaluation of Goal-Oriented Satisfaction Analysis Techniques". Em: *Requirements Engineering* 18.3 (2012), pp. 199–222.

[HYY11] J. Horkoff, Y. Yu e E. Yu. "OpenOME: An Open-source Goal and Agent-Oriented Model Drawing and Analysis Tool". Em: *Proceedings of the 5th International i* Workshop*. Vol. 766. CEUR Workshop Proceedings. 2011, pp. 154–156.

[IEE14] IEEE. *IEEE Stantard for a Software Quality Metrics Methodology*. Último acesso em Setembro de 2014. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=749159.

[jUC14] jUCMNav. *jUCMNav: Juice up your modelling!* Último acesso em Setembro de 2014. URL: http://jucmnav.softwareengineering.ca/ucm/bin/view/ProjetSEG/WebHome.

[Kas03] J. E. Kasser. "Object-Oriented Requirements Engineering and Management". Em: *Systems Engineering Test and Evaluation Conference (SETE 2003)*. Camberra, Austrália, 2003.

[Kol+06] M. Kolp, S. Faulkner, Y. Wautelet, T. Nguyen, A. Coyette, Y. Achbany, H. Hoang e T. Do. "DesCARTES Architect. Business-Driven Software Desing". Em: *Forum des Recherches Doctorales*. Institut d'administration et de gestion, 2006.

[KGM02] M. Kolp, P. Giorgini e J. Mylopoulos. "Information Systems Development Through Social Structures". Em: *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering*. SEKE '02. Ischia, Itália: Association for Computer Machinery, 2002, pp. 183–190. ISBN: 1-58113-556-4.

[KS04] G. Kotonya e I. Sommerville. *Requirements Engineering: Processes and Techniques*. Hoboken, EUA: John Wiley & Sons, Inc., 2004. ISBN: 978-0-471-97208-2.

[KB07] R. Kumar e V. Bhattacherjee. "Applicability of Weyuker Property 9 to Object-Oriented Inheritance Tree Metric - A Discussion". Em: *10th International Conference on Information Technology*. IEEE Computer Society, 2007, pp. 234–236.

[Lam01] A. van Lamsweerde. "Goal-Oriented Requirements Engineering: A Guided Tour". Em: *Proceedings of the 5th IEEE International Symposium on Requirements Engineering*. Washington D.C., EUA: IEEE Computer Society, 2001, pp. 249–262.

[Lam09] A. van Lamsweerde. *Requirements Engineering. From System Goals to UML Models to Software Specifications*. 1ª ed. Hoboken, EUA: John Wiley & Sons, Inc., 2009. ISBN: 978-0470012703.

[Lam14] A. van Lamsweerde. *Goal-Driven Requirements Engineering: the KAOS Approach*. Último acesso em Setembro de 2014. URL: http://www.info.ucl.ac.be/~avl/ReqEng.html.

[Lei+05] J. Leite, Y. Yu, L. Liu, E. Yu e J. Mylopoulos. "Quality-based Software Reuse". Em: *International Conference on Advanced Information Systems Engineering (CAiSE '05)*. Vol. 3520. Lecture Notes in Computer Science. Porto, Portugal: Springer, 13–17 de jun. de 2005, pp. 535–550.

[Lim+11] C. Lima, J. Paes, A. Rodovalho, D. Dermeval e A. Buarque. *MyCourses – A Course Scheduling System*. Centro de Informática, Universidade Federal de Pernambuco, Brasil. SCORE Contest, 2011.

[LY04] L. Liu e E. Yu. "Designing Information Systems in Social Context: A Goal and Scenario Modelling Approach". Em: *Information Systems* 29.2 (2004), pp. 187–203. ISSN: 0306-4379.

[Loc+12] J. Lockerbie, N. A. M. Maiden, J. Engmann, D. Randall, S. Jones e D. Bush. "Exploring the impact of software requirements on system-wide goals: a method using satisfaction arguments and i* goal modelling". Em: *Requirements Engineering* 17 (2012), pp. 227–254.

[Luc+08] M. Lucena, E. Santos, C. Silva, F. Alencar, M. J. Silva e J. Castro. "Towards a Unified Metamodel for i*". Em: IEEE Computer Society, 2008, pp. 237–246. ISBN: 978-1-4244-1677-6.

[MR13] S. Mal e K. Rajnish. "Applicability of Weyuker's Property 9 to Inheritance Metric". Em: *International Journal of Computer Applications* 66.12 (2013), pp. 21–26.

[MA08] S. Misra e I. Akman. "Applicability of Weyuker's properties on OO metrics: Some Misunderstandings". Em: *Computer Science and Information Systems* 5.1 (2008), pp. 17–23.

[Mon10] R. Monteiro. "Engenharia de Requisitos Orientada a Modelos para Abordagens Orientadas a Objectivos". Tese de mestrado. Portugal: Departamento de Informática, Universidade Nova de Lisboa, 2010.

[Mul79] G. P. Mullery. "CORE - A Method for Controlled Requirement Specification". Em: *Proceedings of the 4th International Conference on Software Engineering (ICSE'79)*. Munique, Alemanha: IEEE Press, 1979, pp. 126–135.

[Nat14] National Air Traffic Services. *NATS | A Global Leader in Air Traffic Control and Airport Performance*. Último acesso em Setembro de 2014. URL: http://www.nats.aero/.

[Nun09] C. Nunes. "Uma Linguagem para Domínio Específico para a Framework i*". Tese de mestrado. Portugal: Departamento de Informática, Universidade Nova de Lisboa, 2009.

[NE00] B. Nuseibeh e S. Easterbrook. "Requirements Engineering: a Roadmap". Em: *Proceedings of the 22nd International Conference on Software Engineering*. Limerick, Irlanda: Association for Computing Machinery, 2000, pp. 35–46.

[Obj14a] Object Management Group. *Object Constraint Language (OCL)*. Último acesso em Setembro de 2014. URL: http://www.omg.org/spec/OCL/.

[Obj14b] Object Management Group. *OMG Unified Modeling Language (OMG UML), Infrastructure, V2.1.2*. Último acesso em Setembro de 2014. URL: http://www.omg.org/spec/UML/2.1.2/Infrastructure/PDF/.

[Ope14] OpenOME. *OpenOME, an open-source requirements engineering tool*. Último acesso em Setembro de 2014. URL: https://se.cs.toronto.edu/trac/ome/.

[Pre09] R. Pressman. *Software Engineering: A Practitioner's Approach*. 7ª ed. Nova Iorque, EUA: McGraw-Hill, 2009. ISBN: 978-0073375977.

[Ram+11] R. Ramos, J. Castro, J. Araújo e F. Alencar. "Towards the Improvement of Use Case Models: The AIRDoc Process". Em: *Proceedings of the 2011 ACM Symposium on Applied Computing*. SAC '11. TaiChung, Taiwan: Association for Computing Machinery, 2011, pp. 708–709. ISBN: 978-1-4503-0113-8.

[Ram+08] R. Ramos, J. Castro, J. Araújo, A. Moreira e F. Alencar. "AIRDoc – An Approach to Improve Requirements Documents". Em: *22th Brazilian Symposium on Software Engineering*. SBES, out. de 2008.

[Res14a] Respect-IT. *A KAOS Tutorial, version 1.0 (Oct. 2007)*. Último acesso em Setembro de 2014. URL: http://www.objectiver.com/fileadmin/download/documents/KaosTutorial.pdf.

[Res14b] Respect-IT. *Objectiver*. Último acesso em Setembro de 2014. URL: http://www.objectiver.com/.

[Ros+08] L. M. Rose, R. F. Paige, D. S. Kolovos e F. A. Polack. "The Epsilon Generation Language". Em: *Proceedings of the 4th European Conference on Model Driven Architecture: Foundations and Applications*. ECMDA-FA '08. Berlin, Germany: Springer-Verlag, 2008, pp. 1–16. ISBN: 978-3-540-69095-5.

[RKA06] J.-F. Roy, J. Kealey e D. Amyot. "Towards Integrated Tool Support for the User Requirements Notation". Em: *Proceeding of the 5th International Workshop on System Analysis and Modeling: Language Profiles*. Vol. 4320. Lecture Notes in Computer Science. Springer-Verlag, 2006, pp. 198–215. ISBN: 3-540-68371-2.

[Roy70] W. W. Royce. "Managing the Development of Large Software Systems: Concepts and Techniques". Em: *Proceedings of the IEEE WESTCON*. Reprinted in Proceedings of the 9th International Conference on Software Engineering (ICSE 1989). IEEE Computer Society Press, 1970.

[SSV96] P. Sawyer, I. Sommerville e S. Viller. *PREview: Tackling the Real Concerns of Requirements Engineering*. Rel. téc. Cooperative Systems Engineering Group, Lancaster University, 1996.

[Sch06] D. C. Schmidt. "Guest Editor's Introduction: Model-Driven Engineering". Em: *IEEE Computer* 39.2 (2006), pp. 25–31.

[SCO14] SCORE Contest. *SCORE – Student Contest on Software Engineering*. Último acesso em Setembro de 2014. URL: http://score-contest.org/.

[Shu+02] F. Shull, V. Basili, J. Carver, J. C. Maldonado, G. H. Travassos, M. Mendonça e S. Fabbri. "Replicating Software Engineering Experiments: Addressing the Tacit Knowledge Problem". Em: *Proceedings of the 2002 International Symposium on Empirical Software Engineering (ISESE'02)*. IEEE Computer Society, 2002, pp. 7–16. ISBN: 0-7695-1796-X.

[Sil+05] C. Silva, J. Castro, P. Tedesco e I. Silva. "Describing Agent-Oriented Design Patterns in Tropos". Em: *Proceedings of the 19th Brazilian Symposium in Software Engineering*. Uberlandia, Minas Gerais, Brazil, 2005, pp. 27–78.

[Som10] I. Sommerville. *Software Engineering*. 9ª ed. EUA: Addison-Wesley, 2010. ISBN: 978-0-13-703515-1.

[Som+97] I. Sommerville, I. Sommerville, P. Sawyer e P. Sawyer. "Viewpoints: principles, problems and a practical approach to requirements engineering". Em: *Annals of Software Engineering* 3 (1997), pp. 101–130.

[Sou+12] C. Souza, F. Alencar, G. Guedes, M. Soares, C. Souza, R. Ramos e J. Castro. "AIRDoc-i*: Um Processo para Avaliação de Modelos i*". Em: *WER12 - Workshop em Engenharia de Requisitos*. Buenos Aires, Argentina, 2012. ISBN: 978-987-1635-46-7.

[Sou+13] C. Souza, C. Souza, F. Alencar, J. Castro, E. Figueiredo e P. Cavalcanti. "Avaliação de Modelos i* com o Processo AIRDoc-i*". Em: *Proceedings of Requirements Engineering 2013*. Vol. 1005. CEUR Workshop Proceedings. Rio de Janeiro, Brasil: CEUR-WS.org, 2013.

[STS14] STS-Tool. *Socio Technical Security Modelling Language Tool*. Último acesso em Setembro de 2014. URL: http://www.sts-tool.eu/.

[TAO14] TAOM4E. *Taom4e*. Último acesso em Setembro de 2014. URL: http://selab.fbk.eu/taom/.

[TC99] Q. T. Tran e L. Chung. "Tool Support for Dealing with Non-Functional Requirements". Em: *IEEE Symposium on Application-Specific Systems and Software Engineering Technology*. 1999.

[Tro14a] Tropos Project. *GR-Tool*. Último acesso em Setembro de 2014. URL: http://troposproject.org/tools/grtool/.

[Tro14b] Tropos Project. *Tropos*. Último acesso em Setembro de 2014. URL: http://www.troposproject.org/.

[Vas+11] A. M. L. de Vasconcelos, G. Giachetti, B. Marín e Ó. Pastor. "Towards a CMMI-Compliant Goal-Oriented Software Process through Model-Driven Development". Em: *PoEM*. Vol. 92. Lecture Notes in Business Information Processing. Berlim, Heidelberg: Springer-Verlag, 2011, pp. 253–267. ISBN: 978-3-642-24848-1.

[Vas+12] A. M. L. de Vasconcelos, J. L. de la Vara, J. Sanchez e Ó. Pastor. "Towards CMMI-compliant Business Process-Driven Requirements Engineering". Em: *Proceedings of the 2012 Eighth International Conference on the Quality of Information and Communications Technology*. QUATIC '12. Washington DC, EUA: IEEE Computer Society, 2012, pp. 193–198. ISBN: 978-0-7695-4777-0.

[Vog13] L. Vogel. *Eclipse 4 RCP: The complete guide to Eclipse application development*. Lars Vogel, 2013. ISBN: 978-3943747072.

[W3C14] W3C - World Wide Web Consortium. *Extensible Markup Language (XML)*. Último acesso em Setembro de 2014. URL: http://www.w3.org/XML//.

[WK99] J. Warmer e A. Kleppe. *The Object Constraint Language: Precise Modeling with UML*. Reading, EUA: Addison-Wesley, 1999. ISBN: 978-0-201-37940-2.

[WAC05] I. Webster, J. Amaral e L. M. Cyneiros. "A Survey of Good Practices and Misuses for Modelling with i* Framework". Em: *Proceedings of VIII Workshop on Requirements Engineering*. Porto, Portugal, 2005, pp. 148–160.

[Wey88] E. J. Weyuker. "Evaluating Software Complexity Measures". Em: *IEEE Transactions on Software Engineering* 14 (1988), pp. 1357–1365.

[wik14a] i* wiki. *Comparing the i* Tools*. Último acesso em Setembro de 2014. URL: http://istar.rwth-aachen.de/tiki-index.php?page=Comparing+the+i*+Tools.

[wik14b] i* wiki. *Guideline (Intermediate,Layout) Use the specialized actors notation to the degree that you can gain advantage in instantiating the actual stakeholders.* Último acesso em Setembro de 2014. URL: http://istar.rwth-aachen.de/tiki-index.php?page=Guideline+(Intermediate,Layout)+Use+the+specialized+actors+notation+to+the+degree+

that+you+can+gain+advantage+in+instantiating+the+actual+stakeholders.&structure=i*+Guide.

[wik14c] i* wiki. *i* Guide*. Último acesso em Setembro de 2014. URL: http://istar.rwth-aachen.de/tiki-index.php?page=i*+Guide.

[wik14d] i* wiki. *i* Tools*. Último acesso em Setembro de 2014. URL: http://istar.rwth-aachen.de/tiki-index.php?page=i*+Tools.

[wik14e] i* wiki. *i* wiki*. Último acesso em Setembro de 2014. URL: http://istarwiki.org/.

[wik14f] i* wiki. *Who is Who*. Último acesso em Setembro de 2014. URL: http://istar.rwth-aachen.de/tiki-index.php?page=Who+is+Who&structure=i*+Wiki+Home.

[You05] T. Young. "Using AspectJ to Build a Software Product Line for Mobile Devices". Tese de mestrado. Canadá: University of British Columbia, 2005.

[Yu95] E. Yu. "Modelling Strategic Relationships for Process Reengineering". Tese de doutoramento. Canadá: Department of Computer Science, University of Toronto, 1995.

[Yu97] E. Yu. "Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering". Em: *3rd IEEE International Symposium on Requirements Engineering*. 1997.

[Yu01] E. Yu. "Agent Orientation as a Modelling Paradigm". Em: *Wirtschaftsinformatik* 43.2 (2001), pp. 123–132.

[YY14] E. Yu e Y. Yu. *Organization Modelling Environment*. Último acesso em Setembro de 2014. URL: http://www.cs.toronto.edu/km/ome/.

Meta-modelo

Neste anexo é apresentado o meta-modelo completo do *framework i**. A explicação da maior parte das classes presentes no meta-modelo podem ser encontradas na secção 4.2.

A ligações de dependência (`DependencyLink`) foram separadas em 3 (três) tipos distintos, que permitem ligações diferentes, sendo eles: `DependeeLink`, `DepElemLink` e `DependerLink`. A ligação `DependeeLink` permite realizar ligações entre um *dependum* e um *dependee*. A ligação `DependerLink` permite realizar ligações entre um *depender* e um *dependum*. Por fim, a ligação `DepElemLink` permite realizar ligações entre 2 (dois) elementos. A separação tornou-se necessária por forma a ser possível ter ligações de dependência modeladas de forma correta entre elementos presentes dentro da fronteira de atores.

As ligações de contribuição (`ContributionLink`) podem ser de 9 (nove) tipos diferentes, estando todos representados. Por forma a serem identificáveis, as ligações de contribuição têm um nome.

Por fim, as ligações de associação (`Association`) podem ser de 6 (seis) tipos diferentes, estando todos representados. Por forma a serem identificáveis, as ligações de associação têm um nome.

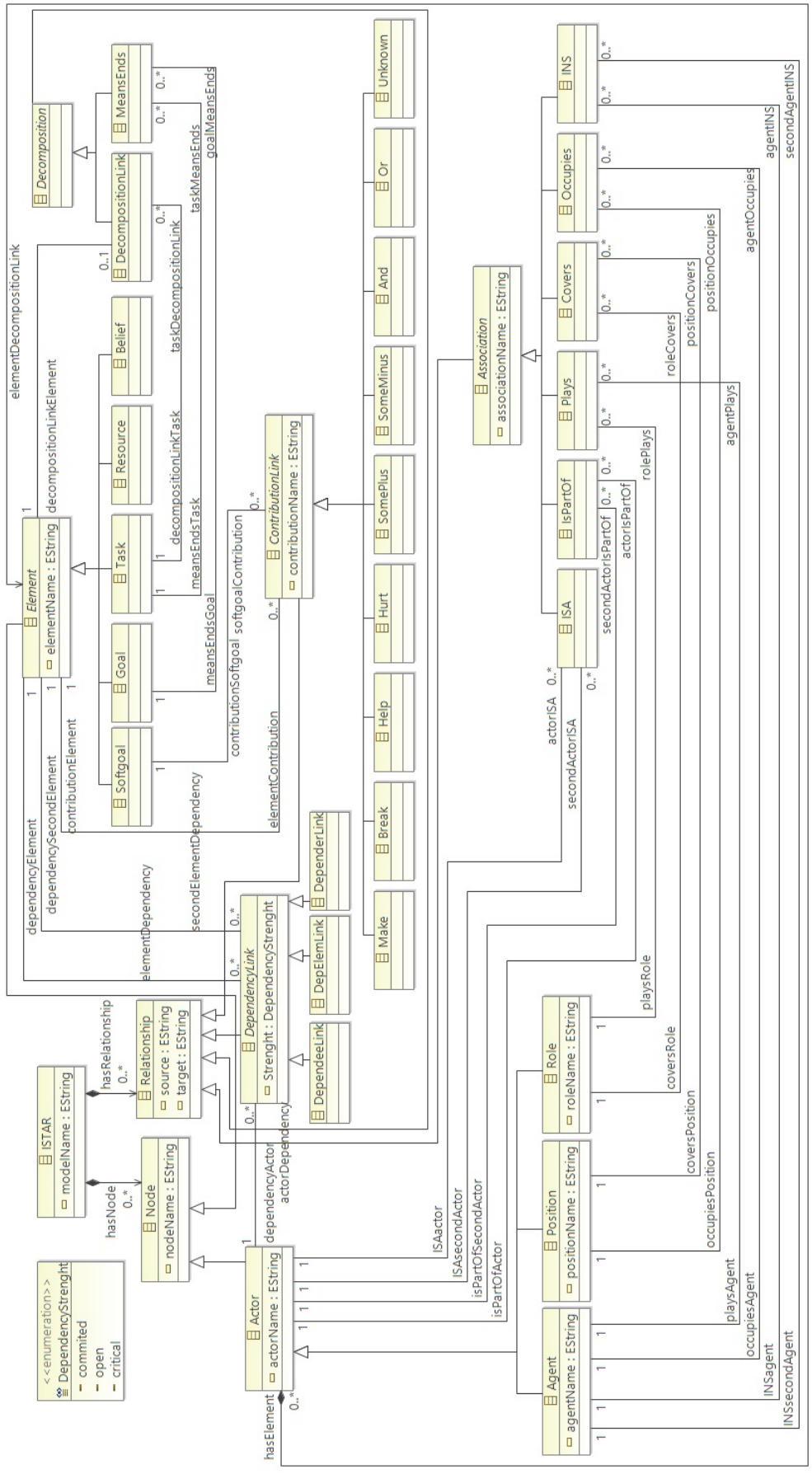


Figura A.1: Meta-modelo completo

B

Manual do Utilizador

Neste anexo é apresentado o manual do utilizador da ferramenta. Começa por definir quais os requisitos do sistema necessários para que a ferramenta funcione corretamente. Depois, é fornecida informação sobre a forma como se deve instalar e executar o projeto onde se encontra desenvolvida a ferramenta. Por fim, é descrito como pode ser criado um novo projeto utilizando a ferramenta, e começar a criação de modelos *i**.

B.1 Requisitos do Sistema

A ferramenta foi desenvolvida utilizando o IDE Eclipse Modeling Tools, versão Kepler Service Release 1. A figura B.1 mostra os *plugins* e respetivas versões que foram utilizados para o desenvolvimento da ferramenta.

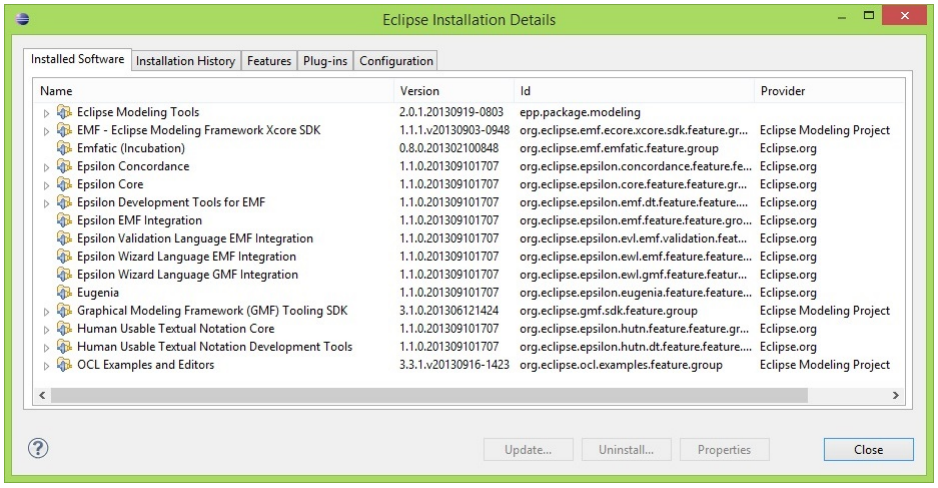


Figura B.1: *Plugins* necessário para a utilização da ferramenta

Os *plugins* podem ser encontrados e instalados através do próprio IDE Eclipse, em *Help* → *Install Modeling Components*, ou através da página *web* de cada um dos *plugins*.

B.2 Instalação e Execução

Antes de se instalar a ferramenta, é necessário verificar se a versão do compilador Java é a 1.7. Para isso, no IDE Eclipse vai-se a *Window* → *Preferences* → *Java* → *Compiler* → *Compiler compliance level*.

Depois desta verificação, e garantindo que os *plugins* estão todos corretamente instalados, faz-se a importação do projeto da ferramenta. Para tal, no IDE Eclipse faz-se *File* → *Import* → *General* → *Existing Projects into Workspace* → *Select root directory* e seleciona-se o caminho para a diretoria que contém o projeto. Selecionam-se as pastas correspondentes e carrega-se em *Finish*.

Na pasta `PrototypeiStar`, em `metamodel`, é possível observar todos os ficheiros necessários para a criação do projeto, como se pode verificar na figura B.2.

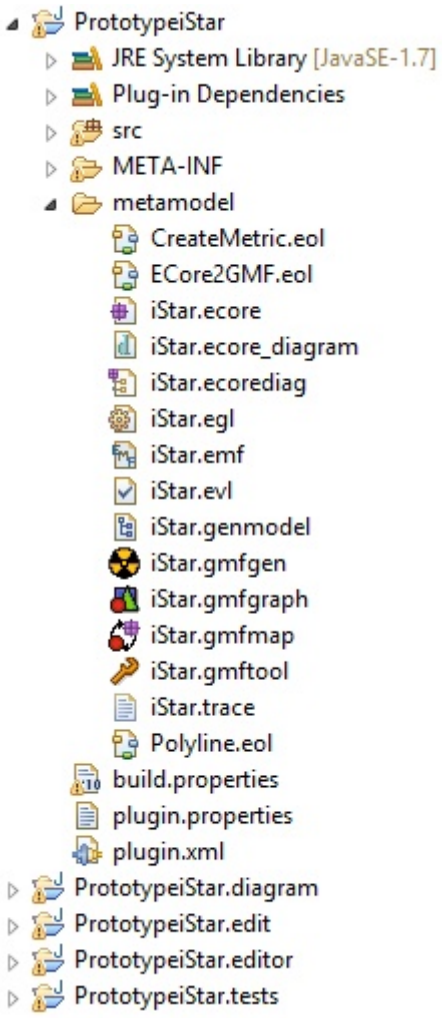


Figura B.2: Sistema de ficheiros do projeto da ferramenta

Para que o projeto corra sem problemas, é necessário ativar o menu sensível ao contexto da pasta `PrototypeiStar` e ir a *Run As* → *Run Configurations...* → *Eclipse Applications* e no separador *Arguments*, em *VM arguments* colocar o seguinte:

```
-Dosgi.requiredJavaVersion=1.7 -Xms256m -Xmx1024m -XX:MaxPermSize=256m
```

Depois disto, para executar a ferramenta é apenas necessário ativar o menu sensível ao contexto da pasta `PrototypeiStar` e ir a *Run As* → *Eclipse Application*.

B.3 Criação de um Projeto

Para criar um projeto com a ferramenta, e depois de a mesma estar em execução, é necessário ir a *File* → *New* → *Project...* → *General* → *Project*. Carrega-se em *Next*, insere-se o nome do projeto e carrega-se em *Finish*.

Uma vez o projeto criado, tem que se criar o editor. Para tal, ativa-se o menu sensível ao contexto do projeto previamente criado e vai-se a *New* → *Example...* e seleciona-se *Istar Diagram*. Depois de se carregar *Next*, adiciona-se o nome do modelo e carrega-se em *Finish*. Dentro da pasta surgem 2 (dois) ficheiros, um com a extensão `.istar` e outro com a extensão `.istar_diagram`. A criação do modelo é feita no segundo ficheiro, e a informação do que for inserido no modelo será apresentada textualmente no primeiro, de forma automática.

O editor gráfico é composto por uma tela, onde o modelo é desenhado, e por uma palete com os elementos do lado direito. Para a apresentação das métricas, seleciona-se o ficheiro `istar_diagram` e vai-se a *Edit* → *Metrics*. Para validar o modelo, o processo é o mesmo mas vai-se a *Edit* → *Validate*.

C

Métricas Auxiliares

Neste anexo são apresentadas as métricas auxiliares, necessárias tanto para o cálculo correto das métricas principais, como para o cálculo de outras métricas auxiliares. Algumas das métricas auxiliares servem a mais do que uma questão, pelo que cada métrica é apresentada de forma individual, pela ordem em que é referida no texto presente na secção 4.3. Nos casos em que as métricas auxiliares necessitem de outras métricas, estas irão estar definidas logo de seguida, sempre que não tenham sido definidas anteriormente. Para cada métrica é apresentado o seu nome, uma definição informal, em linguagem natural, e uma definição formal, em OCL.

C.1 Métricas Auxiliares para a Complexidade

Tabela C.1: Métrica Auxiliar NEOAB

Nome	**NEOAB** – *Number of Elements Outside Actors' Boundaries*
Definição informal	Número total de elementos fora da fronteira de atores no modelo SD/SR

Definição formal	`context ISTAR`
	`def:NEOAB():Integer = self.hasNode ->`
	`  select(n:Node | n.oclIsKindOf(Element)) -> size()`

Tabela C.2: Métrica Auxiliar NEI

Nome	**NEI** – ***N**umber of **E**lements **I**nside*
Definição informal	Número de elementos dentro da fronteira de um ator no modelo SR

Definição formal	`context Actor`
	`def:NEI():Integer = self.hasElement ->`
	`  select(e:Element | e.oclIsKindOf(Element)) -> size()`

Tabela C.3: Métrica Auxiliar NDGI

Nome	**NDGI** – ***N**umber of **D**ecompositions of a **G**oal **I**nside*
Definição informal	Número total de decomposições (*means-end*) associadas a um objetivo dentro da fronteira de um ator no modelo SR

Definição formal	`context Actor`
	`def:NDGI():Integer = self.hasElement ->`
	`  select(e:Element | e.oclIsKindOf(Goal)) ->`
	`  iterate(e:Element; total:Integer = 0 |`
	`  let ndg:Integer = e.oclAsType(Goal).NDG() in total + ndg)`

Necessita	**NDG** – ***N**umber of **D**ecompositions of a **G**oal* (MA C.4)

Tabela C.4: Métrica Auxiliar NDG

Nome	**NDG** – ***N**umber of **D**ecompositions of a **G**oal*
Definição informal	Número de decomposições de um objetivo no modelo SR

Definição formal	`context Goal`
	`def:NDG():Integer = self.goalMeansEnds ->`
	`  select(me: MeansEnds | me.oclIsKindOf(MeansEnds)) -> size()`

Tabela C.5: Métrica Auxiliar MinNDGI

Nome	**MinNDGI** – ***Min**imum **N**umber of **D**ecompositions of a **G**oal **I**nside*
Definição informal	Número mínimo de decomposições associadas a um objetivo dentro da fronteira de um ator no modelo SR

Definição formal	`context Actor`
	`def:MinNDGI():Integer = self.hasElement ->`
	`  select(e:Element | e.oclIsKindOf(Goal) and`
	`  e.oclAsType(Goal).NDG() > 0) ->`
	`  iterate(e:Element; min:Integer = -1 |`
	`  let ndg:Integer = e.oclAsType(Goal).NDG() in`
	`  if min = -1 then ndg else min.min(ndg) endif)`

Necessita	**NDG** – ***N**umber of **D**ecompositions of a **G**oal* (MA C.4)

Tabela C.6: Métrica Auxiliar MaxNDGI

Nome	**MaxNDGI** – ***Max**imum **N**umber of **D**ecompositions of a **G**oal **I**nside*
Definição informal	Número máximo de decomposições associadas a um objetivo dentro da fronteira de um ator no modelo SR

Definição formal	`context Actor`
	`def:MaxNDGI():Integer = self.hasElement ->`
	`  select(e:Element | e.oclIsKindOf(Goal) and`
	`  e.oclAsType(Goal).NDG() > 0) ->`
	`  iterate(e:Element; max:Integer = -1 |`
	`  let ndg:Integer = e.oclAsType(Goal).NDG() in`
	`  if max = -1 then ndg else max.max(ndg) endif)`

Necessita	**NDG** – ***N**umber of **D**ecompositions of a **G**oal* (MA C.4)

Tabela C.7: Métrica Auxiliar NGWD

Nome	**NGWD** – ***N**umber of **G**oals **W**ith **D**ecompositions*
Definição informal	Número total de objetivos com decomposições no modelo SR

Definição formal	`context ISTAR`
	`def:NGWD():Integer = self.hasNode ->`
	`  select(n:Node | n.oclIsKindOf(Actor)) ->`
	`  iterate(n:Node; total:Integer = 0 |`
	`  let ngwdi:Integer = n.oclAsType(Actor).NGWDI() in`
	`  total + ngwdi)`

Necessita	**NGWDI** – ***N**umber of **G**oals **W**ith **D**ecompositions **I**nside* (MA C.8)

Tabela C.8: Métrica Auxiliar NGWDI

Nome	**NGWDI** – ***N**umber of **G**oals **W**ith **D**ecompositions **I**nside*
Definição informal	Número de objetivos com decomposições dentro da fronteira de um ator no modelo SR

Definição formal	`context Actor`
	`def:NGWDI():Integer = self.hasElement ->`
	`  select(e:Element | e.oclIsKindOf(Goal) and`
	`  e.oclAsType(Goal).NDG() > 0) -> size()`

Necessita	**NDG** – ***N**umber of **D**ecompositions of a **G**oal* (MA C.4)

Tabela C.9: Métrica Auxiliar NDSI

Nome	**NDSI** – ***N**umber of **D**ecompositions of a **S**oftgoal **I**nside*
Definição informal	Número total de decomposições associadas a um *softgoal* dentro da fronteira de um ator no modelo SR

Definição formal	`context Actor`
	`def:NDSI():Integer = self.hasElement ->`
	`  select(e:Element | e.oclIsKindOf(Softgoal)) ->`
	`  iterate(e:Element; total:Integer = 0 |`
	`  let nds:Integer = e.oclAsType(Softgoal).NDS() in total + nds)`

Necessita	**NDS** – ***N**umber of **D**ecompositions of a **S**oftgoal* (MA C.10)

Tabela C.10: Métrica Auxiliar NDS

Nome	**NDS** – ***N**umber of **D**ecompositions of a **S**oftgoal*
Definição informal	Número de decomposições de um *softgoal* no modelo SR

Definição formal	`context Softgoal`
	`def:NDS():Integer = self.softgoalContribution ->`
	`  select(cl:ContributionLink |`
	`  cl.oclIsKindOf(ContributionLink)) -> size()`

Tabela C.11: Métrica Auxiliar MinNDSI

Nome	**MinNDSI** – ***M**inimum **N**umber of **D**ecompositions of a **S**oftgoal **I**nside*
Definição informal	Número mínimo de decomposições associadas a um *softgoal* dentro da fronteira de um ator no modelo SR

Definição formal	`context Actor`
	`def:MinNDSI():Integer = self.hasElement ->`
	`  select(e:Element | e.oclIsKindOf(Softgoal) and`
	`  e.oclAsType(Softgoal).NDS() > 0) ->`
	`  iterate(e Element; min:Integer = -1 |`
	`  let nds:Integer = e.oclAsType(Softgoal).NDS() in`
	`  if min = -1 then nds else min.min(nds) endif)`

Necessita	**NDS** – ***N**umber of **D**ecompositions of a **S**oftgoal* (MA C.10)

Tabela C.12: Métrica Auxiliar MaxNDSI

Nome	**MaxNDSI** – ***M**aximum **N**umber of **D**ecompositions of a **S**oftgoal **I**nside*
Definição informal	Número máximo de decomposições associadas a um *softgoal* dentro da fronteira de um ator no modelo SR

Definição formal	`context Actor`
	`def:MaxNDSI():Integer = self.hasElement ->`
	`  select(e:Element | e.oclIsKindOf(Softgoal) and`
	`  e.oclAsType(Softgoal).NDS() > 0) ->`
	`  iterate(e : Element; max : Integer = -1 |`
	`  let nds : Integer = e.oclAsType(Softgoal).NDS() in`
	`  if max = -1 then nds else max.max(nds) endif)`

Necessita	**NDS** – ***N**umber of **D**ecompositions of a **S**oftgoal* (MA C.10)

Tabela C.13: Métrica Auxiliar NSWD

Nome	**NSWD** – ***N**umber of **S**oftgoals **W**ith **D**ecompositions*
Definição informal	Número total de *softgoals* com decomposições no modelo SR

Definição formal	`context ISTAR`
	`def:NSWD():Integer = self.hasNode ->`
	`  select(n:Node | n.oclIsKindOf(Actor)) ->`
	`  iterate(n:Node; total:Integer = 0 |`
	`  let nswdi:Integer = n.oclAsType(Actor).NSWDI() in`
	`  total + nswdi)`

Necessita	**NSWDI** – ***N**umber of **S**oftgoals **W**ith **D**ecompositions **I**nside* (MA C.14)

Tabela C.14: Métrica Auxiliar NSWDI

Nome	**NSWDI** – ***N**umber of **S**oftgoals **W**ith **D**ecompositions **I**nside*
Definição informal	Número de *softgoals* com decomposições dentro da fronteira de um ator no modelo SR

Definição formal	`context Actor`
	`def:NSWDI():Integer = self.hasElement ->`
	`  select(e:Element | e.oclIsKindOf(Softgoal) and`
	`  e.oclAsType(Softgoal).NDS() > 0) -> size()`

Necessita	**NDS** – ***N**umber of **D**ecompositions of a **S**oftgoal* (MA C.10)

Tabela C.15: Métrica Auxiliar NDTI

Nome	**NDTI** – ***N**umber of **D**ecompositions of a **T**ask **I**nside*
Definição informal	Número total de decomposições associadas a uma tarefa dentro da fronteira de um ator no modelo SR

Definição formal	`context Actor`
	`def:NDTI():Integer = self.hasElement ->`
	`  select(e:Element | e.oclIsKindOf(Task)) ->`
	`  iterate(e:Element; total:Integer = 0 |`
	`  let ndt:Integer = e.oclAsType(Task).NDT() in total + ndt)`

Necessita	**NDT** – ***N**umber of **D**ecompositions of a **T**ask* (MA C.16)

Tabela C.16: Métrica Auxiliar NDT

Nome	**NDT** – ***N**umber of **D**ecompositions of a **T**ask*
Definição informal	Número de decomposições de uma tarefa no modelo SR

Definição formal	`context Task`
	`def:NDT():Integer = self.taskDecompositionLink ->`
	`  select(dl : DecompositionLink |`
	`  dl.oclIsKindOf(DecompositionLink)) -> size()`

Tabela C.17: Métrica Auxiliar MinNDTI

Nome	**MinNDTI** – ***Min**imum **N**umber of **D**ecompositions of a **T**ask **I**nside*
Definição informal	Número mínimo de decomposições associadas a uma tarefa dentro da fronteira de um ator no modelo SR

Definição formal	`context Actor`
	`def:MinNDTI():Integer = self.hasElement ->`
	`  select(e:Element | e.oclIsKindOf(Task) and`
	`  e.oclAsType(Task).NDT() > 0) ->`
	`  iterate(e:Element; min:Integer = -1 |`
	`  let ndt:Integer = e.oclAsType(Task).NDT() in`
	`  if min = -1 then ndt else min.min(ndt) endif)`

Necessita	**NDT** – ***N**umber of **D**ecompositions of a **T**ask* (MA C.16)

Tabela C.18: Métrica Auxiliar MaxNDTI

Nome	**MaxNDTI** – ***Ma**ximum **N**umber of **D**ecompositions of a **T**ask **I**nside*
Definição informal	Número máximo de decomposições associadas a uma tarefa dentro da fronteira de um ator no modelo SR

Definição formal	`context Actor`
	`def:MaxNDTI():Integer = self.hasElement ->`
	`  select(e:Element | e.oclIsKindOf(Task) and`
	`  e.oclAsType(Task).NDT() > 0) ->`
	`  iterate(e:Element; max:Integer = -1 |`
	`  let ndt:Integer = e.oclAsType(Task).NDT() in`
	`  if max = -1 then ndt else max.max(ndt) endif)`

Necessita	**NDT** – ***N**umber of **D**ecompositions of a **T**ask* (MA C.16)

Tabela C.19: Métrica Auxiliar NTWD

Nome	**NTWD** – ***N**umber of **T**asks **W**ith **D**ecompositions*
Definição informal	Número total de tarefas com decomposições no modelo SR

Definição formal	`context ISTAR`
	`def:NTWD():Integer = self.hasNode ->`
	`  select(n:Node | n.oclIsKindOf(Actor)) ->`
	`  iterate(n:Node; total:Integer = 0 |`
	`  let ntwdi:Integer = n.oclAsType(Actor).NTWDI() in`
	`  total + ntwdi)`

Necessita	**NTWDI** – ***N**umber of **T**asks **W**ith **D**ecompositions **I**nside* (MA C.20)

Tabela C.20: Métrica Auxiliar NTWDI

Nome	**NTWDI** – ***N**umber of **T**asks **W**ith **D**ecompositions **I**nside*
Definição informal	Número de tarefas com decomposições dentro da fronteira de um ator no modelo SR

Definição formal	`context Actor`
	`def:NTWDI():Integer = self.hasElement ->`
	`  select(e:Element | e.oclIsKindOf(Task) and`
	`  e.oclAsType(Task).NDT() > 0)->size()`

Necessita	**NDT** – ***N**umber of **D**ecompositions of a **T**ask* (MA C.16)

Tabela C.21: Métrica Auxiliar NODA

Nome	**NODA** – ***N**umber of **O**utgoing **D**ependencies of **A**ctors*
Definição informal	Número total de dependências de saída de todos os atores no modelo SD/SR

Definição formal	`context ISTAR`
	`def:NODA():Integer = self.hasNode ->`
	`  select(n:Node | n.oclIsKindOf(Actor)) ->`
	`  iterate(n:Node; total:Integer = 0 |`
	`  let nod:Integer = n.oclAsType(Actor).NOD() in total + nod)`

Necessita	**NOD** – ***N**umber of **O**utgoing **D**ependencies* (MA C.22)

Tabela C.22: Métrica Auxiliar NOD

Nome	**NOD** - *Number of Outgoing Dependencies*
Definição informal	Número de dependências de saída de um ator no modelo SD/SR
Definição formal	`context Actor` `def:NOD():Integer = self.NODAI() + self.NODEI()`
Necessita	**NODAI** – *Number of Outgoing Dependencies of an Actor Itself* (MA C.23) **NODEI** – *Number of Outgoing Dependencies of an Element Inside* (MA C.24)

Tabela C.23: Métrica Auxiliar NODAI

Nome	**NODAI** – *Number of Outgoing Dependencies of an Actor Itself*
Definição informal	Número dependências de saída associadas a um ator no modelo SD/SR

Definição formal	`context Actor`
	`def:NODAI():Integer = self.actorDependency ->`
	`  select(dl:DependencyLink |`
	`  dl.oclIsKindOf(DependerLink)) -> size()`

Tabela C.24: Métrica Auxiliar NODEI

Nome	**NODEI** – *Number of Outgoing Dependencies of an Element Inside*
Definição informal	Número total de dependências de saída de elementos dentro da fronteira de um ator no modelo SR
Definição formal	`context Actor` `def:NODEI():Integer = self.hasElement ->` ` select(e:Element	e.oclIsKindOf(Element)) ->` ` iterate(e:Element; total:Integer = 0 |` ` let nodepe:Integer = e.oclAsType(Element).NODepE() in` ` total + nodepe)`
Necessita	**NODepE** – *Number of Outgoing Dependencies of an Element* (C.25)

Tabela C.25: Métrica Auxiliar NODepE

Nome	**NODepE** – *Number of Outgoing Dependencies of an Element*
Definição informal	Número total de dependências de saída de um elemento no modelo SD/SR
Definição formal	`context Element` `def:NODepE():Integer = self.NODE() + self.NODEA()`
Necessita	**NODE** – *Number of Outgoing Dependencies from an Element* (MA C.26) **NODEA** – *Number of Outgoing Dependencies from an Element to an Actor* (MA C.27)

Tabela C.26: Métrica Auxiliar NODE

Nome	**NODE** – *Number of Outgoing Dependencies from an Element*
Definição informal	Número de dependências de saída de um elemento para um elemento no modelo SD/SR

Definição formal	`context Element`
	`def:NODE():Integer = self.elementDependency ->`
	`  select(dl:DependencyLink |`
	`  dl.oclIsKindOf(DepElemLink)) -> size()`

Tabela C.27: Métrica Auxiliar NODEA

Nome	**NODEA** – *Number of Outgoing Dependencies from an Element to an Actor*
Definição informal	Número de dependências de saída de um elemento para um ator no modelo SD/SR

Definição formal	`context Element`
	`def:NODEA():Integer = self.elementDependency ->`
	`  select(dl:DependencyLink |`
	`  dl.oclIsKindOf(DependeeLink)) -> size()`

Tabela C.28: Métrica Auxiliar NIDA

Nome	**NIDA** – *Number of Incoming Dependencies of Actors*
Definição informal	Número total de dependências de entrada de todos os atores no modelo SD/SR

Definição formal	`context ISTAR`
	`def:NIDA():Integer = self.hasNode ->`
	`  select(n:Node | n.oclIsKindOf(Actor)) ->`
	`  iterate(n:Node; total:Integer = 0 |`
	`  let nid:Integer = n.oclAsType(Actor).NID() in total + nid)`

Necessita	**NID** – *Number of Incoming Dependencies* (MA C.29)

Tabela C.29: Métrica Auxiliar NID

Nome	**NID** - *Number of Incoming Dependencies*
Definição informal	Número de dependências de entrada de um ator no modelo SD/SR
Definição formal	`context Actor` `def:NID():Integer = self.NIDAI() + self.NIDEI()`
Necessita	**NIDAI** – *Number of Incoming Dependencies of an Actor Itself* (MA C.30) **NIDEI** – *Number of Incoming Dependencies of an Element Inside* (MA C.31)

Tabela C.30: Métrica Auxiliar NIDAI

Nome	**NIDAI** – *Number of Incoming Dependencies of an Actor Itself*
Definição informal	Número de dependências de entrada associadas a um ator no modelo SD/SR

Definição formal	`context Actor`
	`def:NIDAI():Integer = self.actorDependency ->`
	`  select(dl:DependencyLink |`
	`  dl.oclIsKindOf(DependeeLink)) -> size()`

Tabela C.31: Métrica Auxiliar NIDEI

Nome	**NIDEI** – *Number of Incoming Dependencies of an Element Inside*
Definição informal	Número total de dependências de entrada de elementos dentro da fronteira de um ator no modelo SD

Definição formal	`context Actor`
	`def:NIDEI():Integer = self.hasElement ->`
	`  select(e:Element | e.oclIsKindOf(Element)) ->`
	`  iterate(e:Element; total:Integer = 0 |`
	`  let nidepe:Integer = e.oclAsType(Element).NIDepE() in`
	`  total + nidepe)`

Necessita	**NIDepE** – *Number of Incoming Dependencies of an Element* (MA C.32)

Tabela C.32: Métrica Auxiliar NIDepE

Nome	**NIDepE** – *Number of Incoming Dependencies of an Element*
Definição informal	Número total de dependências de entrada de um elemento no modelo SD/SR
Definição formal	`context Element` `def:NIDepE():Integer = self.NIDE() + self.NIDEA()`
Necessita	**NIDE** – *Number of Incoming Dependencies of an Element* (MA C.33) **NIDEA** – *Number of Incoming Dependencies to an Element from an Actor* (MA C.34)

Tabela C.33: Métrica Auxiliar NIDE

Nome	**NIDE** – *Number of Incoming Dependencies of an Element*
Definição informal	Número de dependências de entrada de um elemento para um elemento no modelo SD/SR

Definição formal	`context Element`
	`def:NIDE():Integer = self.secondElementDependency ->`
	`  select(dl:DependencyLink |`
	`  dl.oclIsKindOf(DepElemLink)) -> size()`

Tabela C.34: Métrica Auxiliar NIDEA

Nome	**NIDEA** – *Number of Incoming Dependencies to an Element from an Actor*
Definição informal	Número de dependências de entrada para um elemento de um ator no modelo SD/SR

Definição formal	`context Element`
	`def:NIDEA():Integer = self.elementDependency ->`
	`  select(dl:DependencyLink |`
	`  dl.oclIsKindOf(DependerLink)) -> size()`

Tabela C.35: Métrica Auxiliar NDA

Nome	**NDA** – *Number of Dependencies of Actors*
Definição informal	Número total de dependências de todos os atores no modelo SD/SR
Definição formal	`context ISTAR` `def:NDA():Integer = self.NIDA() + self.NODA()`
Necessita	**NIDA** – *Number of Incoming Dependencies of Actors* (MA C.28) **NODA** – *Number of Outgoing Dependencies of Actors* (MA C.21)

Tabela C.36: Métrica Auxiliar ND

Nome	**ND** – *Number of Dependencies*
Definição informal	Número de ligações de dependência de um ator no modelo SD/SR
Definição formal	`context Actor` `def:ND():Integer = self.NID() + self.NOD()`
Necessita	**NID** – *Number of Incoming Dependencies* (MA C.29) **NOD** – *Number of Outgoing Dependencies* (MA C.22)

Tabela C.37: Métrica Auxiliar POD

Nome	**POD** – *Percentage of Outgoing Dependencies*
Definição informal	Percentagem de dependências de saída de um ator no modelo SD/SR

Definição formal	`context Actor::POD()`
	`pre: self.ND() > 0`
	
	`context Actor`
	`def:POD():Double = self.NOD() / self.ND()`

Necessita	**NOD** – *Number of Outgoing Dependencies* (MA C.22) **ND** – *Number of Dependencies* (MA C.36)

Tabela C.38: Métrica Auxiliar PID

Nome	**PID** – ***P**ercentage of **I**ncoming **D**ependencies*
Definição informal	Percentagem de dependências de entrada de um ator no modelo SD/SR

Definição formal	`context Actor::PID()`
	`pre: self.ND() > 0`
	
	`context Actor`
	`def:PID():Double = self.NID() / self.ND()`

Necessita	**NID** – ***N**umber of **I**ncoming **D**ependencies* (MA C.29) **ND** – ***N**umber of **D**ependencies* (MA C.36)

Tabela C.39: Métrica Auxiliar NEIA

Nome	**NEIA** – ***N**umber of **E**lements **I**nside an **A**gent*
Definição informal	Número de elementos dentro da fronteira de um agente

Definição formal	`context Agent`
	`def:NEIA():Integer = self.hasElement ->`
	`  select(e:Element | e.oclIsKindOf(Element)) -> size()`

Tabela C.40: Métrica Auxiliar NEIP

Nome	**NEIP** – ***N**umber of **E**lements **I**nside a **P**osition*
Definição informal	Número de elementos dentro da fronteira de uma posição

Definição formal	`context Position`
	`def:NEIP():Integer = self.hasElement ->`
	`  select(e:Element | e.oclIsKindOf(Element)) -> size()`

Tabela C.41: Métrica Auxiliar NEIR

Nome	**NEIR** – ***N**umber of **E**lements **I**nside a **R**ole*
Definição informal	Número de elementos dentro da fronteira de um papel

Definição formal	`context Role`
	`def:NEIR():Integer = self.hasElement ->`
	`  select(e:Element | e.oclIsKindOf(Element)) -> size()`

C.2 Métricas Auxiliares para a Completude

Tabela C.42: Métrica Auxiliar NAgents

Nome	**NAgents** – ***N**umber of **A**gents*
Definição informal	Número de agentes no modelo SD/SR

Definição formal	`context ISTAR`
	`def:NAgents():Integer = self.hasNode ->`
	`  select(n:Node | n.oclIsKindOf(Agent)) -> size()`

Tabela C.43: Métrica Auxiliar NRoles

Nome	**NRoles** – *Number of **Roles***
Definição informal	Número de papéis no modelo SD/SR

Definição formal	`context ISTAR`
	`def:NRoles():Integer = self.hasNode ->`
	`  select(n:Node | n.oclIsKindOf(Role)) -> size()`

Tabela C.44: Métrica Auxiliar NPos

Nome	**NPos** - *Number of **Pos**itions*
Definição informal	Número de posições no modelo SD/SR

Definição formal	`context ISTAR`
	`def:NPos():Integer = self.hasNode ->`
	`  select(n:Node | n.oclIsKindOf(Position)) -> size()`

Tabela C.45: Métrica Auxiliar NGIAB

Nome	**NGIAB** – *Number of Goals Inside Actors' Boundaries*
Definição informal	Número total de objetivos dentro da fronteira de todos os atores no modelo SR

Definição formal	`context ISTAR`
	`def:NGIAB():Integer = self.hasNode ->`
	`  select(n:Node | n.oclIsKindOf(Actor)) ->`
	`  iterate(n:Node; total:Integer = 0 |`
	`  let ngi:Integer = n.oclAsType(Actor).NGI() in`
	`  total + ngi)`

Necessita	**NGI** – *Number of Goals Inside* (MA C.46)

Tabela C.46: Métrica Auxiliar NGI

Nome	**NGI** – *Number of Goals Inside*
Definição informal	Número de objetivos dentro da fronteira de um ator no modelo SR

Definição formal	`context Actor`
	`def:NGI():Integer = self.hasElement ->`
	`  select(e:Element | e.oclIsKindOf(Goal)) -> size()`

Tabela C.47: Métrica Auxiliar NSIAB

Nome	**NSIAB** – *Number of Softgoals Inside Actors' Boundaries*
Definição informal	Número total de *softgoals* dentro da fronteira de todos os atores no modelo SR

Definição formal	`context ISTAR`
	`def:NSIAB():Integer = self.hasNode ->`
	`  select(n:Node | n.oclIsKindOf(Actor)) ->`
	`  iterate(n:Node; total:Integer = 0 |`
	`  let nsi : Integer = n.oclAsType(Actor).NSI() in total + nsi)`

Necessita	**NSI** – *Number of Softgoals Inside* (C.48)

Tabela C.48: Métrica Auxiliar NSI

Nome	**NSI** – ***N**umber of **S**oftgoals **I**nside*
Definição informal	Número de *softgoals* dentro da fronteira de um ator no modelo SR

Definição formal	`context Actor`
	`def:NSI():Integer = self.hasElement ->`
	`  select(e:Element | e.oclIsKindOf(Softgoal)) -> size()`

Tabela C.49: Métrica Auxiliar NAWEI

Nome	**NAWEI** – ***N**umber of **A**ctors **W**ith **E**lements **I**nside*
Definição informal	Número de atores com elementos dentro da sua fronteira no modelo SR

Definição formal	`context ISTAR`
	`def:NAWEI():Integer = self.hasNode ->`
	`  select(n:Node | n.oclIsKindOf(Actor) and`
	`  n.oclAsType(Actor).NEI() > 0) -> size()`

Necessita	**NEI** – ***N**umber of **E**lements **I**nside* (MA C.2)

Tabela C.50: Métrica Auxiliar PAWUEI

Nome	**PAWUEI** – ***P**ercentage of **A**ctors **W**ith **U**nconnected **E**lements **I**nside*
Definição informal	Percentagem de atores com elementos desconexos dentro da sua fronteira no modelo SR

Definição formal	`context ISTAR::PAWUEI`
	`pre: self.NAct() > 0`
	
	`context ISTAR`
	`def:PAWUEI():Double = self.NAWUEI() / self.NAWEI()`

Necessita	**NAWUEI** – ***N**umber of **A**ctors **W**ith **U**nconnected **E**lements **I**nside* (MA C.51) **NAWEI** – ***N**umber of **A**ctors **W**ith **E**lements **I**nside* (MA C.49)

Tabela C.51: Métrica Auxiliar NAWUEI

Nome	**NAWUEI** – ***N**umber of **A**ctors **W**ith **U**nconnected **E**lements **I**nside*
Definição informal	Número de atores com elementos desconexos dentro da sua fronteira no modelo SR

Definição formal	`context ISTAR`
	`def:NAWUEI():Integer = self.hasNode ->`
	`  select(n:Node | n.oclIsKindOf(Actor) and`
	`  n.oclAsType(Actor).NUEI() > 0) -> size()`

Necessita	**NUEI** – ***N**umber of **U**nconnected **E**lements **I**nside* (MA C.52)

Tabela C.52: Métrica Auxiliar NUEI

Nome	**NUEI** – ***N**umber of **U**nconnected **E**lements **I**nside*
Definição informal	Número de elementos desconexos dentro da fronteira de um ator no modelo SR

Definição formal	`context Actor`
	`def:NUEI():Integer = self.NUGI() + self.NUSI() +`
	`  self.NUTI() + self.NURI() + self.NUBI()`

Necessita	**NUGI** – *Number of **U**nconnected **G**oals **I**nside* (MA C.53) **NUSI** – *Number of **U**nconnected **S**oftgoals **I**nside* (MA C.57) **NUTI** – *Number of **U**nconnected **T**asks **I**nside* (MA C.61) **NURI** – *Number of **U**nconnected **R**esources **I**nside* (MA C.65) **NUBI** – *Number of **U**nconnected **B**eliefs **I**nside* (MA C.69)

Tabela C.53: Métrica Auxiliar NUGI

Nome	**NUGI** – *Number of **U**nconnected **G**oals **I**nside*
Definição informal	Número de objetivos desconexos dentro da fronteira de um ator no modelo SR

Definição formal	`context Actor`
	`def:NUGI():Integer = self.hasElement ->`
	`  select(e:Element | e.oclIsKindOf(Goal) and`
	`  e.oclAsType(Goal).NLG() = 0) -> size()`

Necessita	**NLG** – *Number of **L**inks of a **G**oal* (MA C.54)

Tabela C.54: Métrica Auxiliar NLG

Nome	**NLG** – *Number of **L**inks of a **G**oal*
Definição informal	Número de ligações de um objetivo de um ator no modelo SR
Definição formal	`context Goal` `def:NLG():Integer = self.NDG() + self.NCLG() + self.NDLG()`
Necessita	**NDG** – *Number of **D**ecompositions of a **G**oal* (MA C.4) **NCLG** – *Number of **C**ontribution **L**inks of a **G**oal* (MA C.55) **NDLG** – *Number of **D**ecomposition **L**inks of a **G**oal* (MA C.56)

Tabela C.55: Métrica Auxiliar NCLG

Nome	**NCLG** – *Number of **C**ontribution **L**inks of a **G**oal*
Definição informal	Número total de ligações de contribuição associadas a um objetivo dentro da fronteira de um ator no modelo SR

Definição formal	`context Goal`
	`def:NCLG():Integer = self.elementContribution ->`
	`  select(cl:ContributionLink |`
	`  cl.oclIsKindOf(ContributionLink)) -> size()`

Tabela C.56: Métrica Auxiliar NDLG

Nome	**NDLG** – ***N**umber of **D**ecomposition **L**inks of a **G**oal*
Definição informal	Número total de ligações de decomposição associadas a um objetivo dentro da fronteira de um ator no modelo SR

Definição formal	`context Goal`
	`def:NDLG():Integer = self.elementDecompositionLink ->`
	`  select(dl:DecompositionLink |`
	`  dl.oclIsKindOf(DecompositionLink)) -> size()`

Tabela C.57: Métrica Auxiliar NUSI

Nome	**NUSI** – ***N**umber of **U**nconnected **S**oftgoals **I**nside*
Definição informal	Número de *softgoals* desconexos dentro da fronteira de um ator no modelo SR
Definição formal	`context Actor` `def:NUSI():Integer = self.hasElement ->` ` select(e:Element	e.oclIsKindOf(Softgoal) and` ` e.oclAsType(Softgoal).NLS() = 0) -> size()`
Necessita	**NLS** – ***N**umber of **L**inks of a **S**oftgoal* (MA C.58)

Tabela C.58: Métrica Auxiliar NLS

Nome	**NLS** – ***N**umber of **L**inks of a **S**oftgoal*
Definição informal	Número de ligações de um *softgoal* de um ator no modelo SR
Definição formal	`context Softgoal` `def:NLS():Integer = self.NDS() + self.NCLS() + self.NDLS()`
Necessita	**NDS** – ***N**umber of **D**ecompositions of a **S**oftgoal* (MA C.10) **NCLS** – ***N**umber of **C**ontribution **L**inks of a **S**oftgoal* (MA C.59) **NDLS** – ***N**umber of **D**ecomposition **L**inks of a **S**oftgoal* (MA C.60)

Tabela C.59: Métrica Auxiliar NCLS

Nome	**NCLS** – ***N**umber of **C**ontribution **L**inks of a **S**oftgoal*
Definição informal	Número total de ligações de contribuição associadas a um *softgoal* dentro da fronteira de um ator no modelo SR

Definição formal	`context Softgoal`
	`def:NCLS():Integer = self.elementContribution ->`
	`  select(cl:ContributionLink |`
	`  cl.oclIsKindOf(ContributionLink)) -> size()`

Tabela C.60: Métrica Auxiliar NDLS

Nome	**NDLS** – ***N**umber of **D**ecomposition **L**inks of a **S**oftgoal*
Definição informal	Número total de ligações de decomposição associadas a um *softgoal* dentro da fronteira de um ator no modelo SR

Definição formal	`context Softgoal`
	`def:NDLS():Integer = self.elementDecompositionLink ->`
	`  select(dl:DecompositionLink |`
	`  dl.oclIsKindOf(DecompositionLink)) -> size()`

Tabela C.61: Métrica Auxiliar NUTI

Nome	**NUTI** – ***N**umber of **U**nconnected **T**asks **I**nside*
Definição informal	Número de tarefas desconexas dentro da fronteira de um ator no modelo SR
Definição formal	`context Actor` `def:NUTI():Integer = self.hasElement ->` ` select(e:Element	e.oclIsKindOf(Task) and` ` e.oclAsType(Task).NLT() = 0) -> size()`
Necessita	**NLT** – ***N**umber of **L**inks of a **T**ask* (MA C.62)

Tabela C.62: Métrica Auxiliar NLT

Nome	**NLT** – ***N**umber of **L**inks of a **T**ask*
Definição informal	Número de ligações de uma tarefa de um ator no modelo SR
Definição formal	`context Task` `def:NLT():Integer = self.NDT() + self.NCLT() + self.NDLT()`
Necessita	**NDT** – ***N**umber of **D**ecompositions of a **T**ask* (MA C.16) **NCLT** – ***N**umber of **C**ontribution **L**inks of a **T**ask* (MA C.63) **NDLT** – ***N**umber of **D**ecomposition **L**inks of a **T**ask* (MA C.64)

Tabela C.63: Métrica Auxiliar NCLT

Nome	**NCLT** – ***N**umber of **C**ontribution **L**inks of a **T**ask*
Definição informal	Número total de ligações de contribuição associadas a uma tarefa dentro da fronteira de um ator no modelo SR

Definição formal	`context Task`
	`def:NCLT():Integer = self.elementContribution ->`
	`  select(cl:ContributionLink |`
	`  cl.oclIsKindOf(ContributionLink)) -> size()`

Tabela C.64: Métrica Auxiliar NDLT

Nome	**NDLT** – *Number of Decomposition Links of a Task*
Definição informal	Número total de ligações de decomposição associadas a uma tarefa dentro da fronteira de um ator no modelo SR

Definição formal	`context Task`
	`def:NDLT():Integer = self.elementDecompositionLink ->`
	`  select(dl:DecompositionLink |`
	`  dl.oclIsKindOf(DecompositionLink)) -> size()`

Tabela C.65: Métrica Auxiliar NURI

Nome	**NURI** – *Number of Unconnected Resources Inside*
Definição informal	Número de recursos desconexos dentro da fronteira de um ator no modelo SR
Definição formal	`context Actor` `def:NURI():Integer = self.hasElement ->` ` select(e:Element	e.oclIsKindOf(Resource) and` ` e.oclAsType(Resource).NLR() = 0) -> size()`
Necessita	**NLR** –*Number of Links of a Resource* (MA C.66)

Tabela C.66: Métrica Auxiliar NLR

Nome	**NLR** –*Number of Links of a Resource*
Definição informal	Número de ligações de um recurso de um ator no modelo SR
Definição formal	`context Resource` `def:NLR():Integer = self.NCLR() + self.NDLR()`
Necessita	**NCLR** – *Number of Contribution Links of a Resource* (MA C.67) **NDLR** – *Number of Decomposition Links of a Resource* (MA C.68)

Tabela C.67: Métrica Auxiliar NCLR

Nome	**NCLR** – *Number of Contribution Links of a Resource*
Definição informal	Número total de ligações de contribuição associadas a um recurso dentro da fronteira de um ator no modelo SR

Definição formal	`context Resource`
	`def:NCLR():Integer = self.elementContribution ->`
	`  select(cl:ContributionLink |`
	`  cl.oclIsKindOf(ContributionLink)) -> size()`

Tabela C.68: Métrica Auxiliar NDLR

Nome	**NDLR** – ***N**umber of **D**ecomposition **L**inks of a **R**esource*
Definição informal	Número total de ligações de decomposição associadas a um recurso dentro da fronteira de um ator no modelo SR

Definição formal	`context Resource`
	`def:NDLR():Integer = self.elementDecompositionLink ->`
	`  select(dl:DecompositionLink |`
	`  dl.oclIsKindOf(DecompositionLink)) -> size()`

Tabela C.69: Métrica Auxiliar NUBI

Nome	**NUBI** – ***N**umber of **U**nconnected **B**eliefs **I**nside*
Definição informal	Número de crenças desconexas dentro da fronteira de um ator

Definição formal	`context Actor`
	`def:NUBI():Integer = self.hasElement ->`
	`  select(e:Element | e.oclIsKindOf(Belief) and`
	`  e.oclAsType(Belief).NLB() = 0) -> size()`

Necessita	**NLB** – ***N**umber of **L**inks of a **B**elief* (MA C.70)

Tabela C.70: Métrica Auxiliar NLB

Nome	**NLB** – ***N**umber of **L**inks of a **B**elief*
Definição informal	Número de ligações de uma crença de um ator no modelo SR
Definição formal	`context Belief` `def:NLB():Integer = self.NCLB() + self.NDLB()`
Necessita	**NCLB** – ***N**umber of **C**ontribution **L**inks of a **B**elief* (MA C.71) **NDLB** – ***N**umber of **D**ecomposition **L**inks of a **B**elief* (MA C.72)

Tabela C.71: Métrica Auxiliar NCLB

Nome	**NCLB** – ***N**umber of **C**ontribution **L**inks of a **B**elief*
Definição informal	Número total de ligações de contribuição associadas a uma crença dentro da fronteira de um ator no modelo SR

Definição formal	`context Belief`
	`def:NCLB():Integer = self.elementContribution ->`
	`  select(cl:ContributionLink |`
	`  cl.oclIsKindOf(ContributionLink)) -> size()`

Tabela C.72: Métrica Auxiliar NDLB

Nome	**NDLB** – *Number of Decomposition Links of a Belief*
Definição informal	Número total de ligações de decomposição associadas a uma crença dentro da fronteira de um ator no modelo SR

Definição formal	`context Belief`
	`def:NDLB():Integer = self.elementDecompositionLink ->`
	`  select(dl:DecompositionLink |`
	`  dl.oclIsKindOf(DecompositionLink)) -> size()`

Tabela C.73: Métrica Auxiliar NAWDOA

Nome	**NAWDOA** – *Number of Actors With Dependencies Or Associations*
Definição informal	Número atores com ligações de dependência ou de associação no modelo SD/SR

Definição formal	`context ISTAR`
	`def:NAWDOA():Integer = self.hasNode ->`
	`  select(n:Node | n.oclIsKindOf(Actor) and`
	`  (n.oclAsType(Actor).ND() > 0 or`
	`  n.oclAsType(Actor).NA() > 0)) -> size()`

Necessita	**ND** – *Number of Dependencies* (MA C.36) **NA** – *Number of Associations* (MA C.74)

Tabela C.74: Métrica Auxiliar NA

Nome	**NA** – *Number of Associations*
Definição informal	Número total de ligações de associação de um ator no modelo SD/SR
Definição formal	`context Actor` `def:NA():Integer = self.NISA() + self.NIsPartOf()`
Necessita	**NISA** – *Number of ISA* (MA C.75) **NIsPartOf** – *Number of IsPartOf* (MA C.76)

Tabela C.75: Métrica Auxiliar NISA

Nome	**NISA** – *Number of ISA*
Definição informal	Número total de associações ISA de um ator no modelo SD/SR

Definição formal	`context Actor`
	`def:NISA():Integer = self.actorISA ->`
	`  select(isa:ISA | isa.oclIsKindOf(ISA)) -> size()`

Tabela C.76: Métrica Auxiliar NIsPartOf

Nome	**NIsPartOf** – *Number of IsPartOf*
Definição informal	Número total de associações IsPartOf de um ator no modelo SD/SR

Definição formal	`context Actor`
	`def:NIsPartOf():Integer = self.actorIsPartOf ->`
	`  select(ipo:IsPartOf | ipo.oclIsKindOf(IsPartOf)) -> size()`

C.3 Métricas Auxiliares para a Correção

Tabela C.77: Métrica Auxiliar TNA

Nome	**TNA** – ***T**otal **N**umber of **A**ssociations*
Definição informal	Número total de ligações de associação no modelo SD/SR

Definição formal	`context ISTAR`
	`def:TNA():Integer = self.hasNode ->`
	`  select(n:Node | n.oclIsKindOf(Actor)) ->`
	`  iterate(n:Node; total:Integer = 0 |`
	`  let na:Integer = n.oclAsType(Actor).NA() in total + na)`

Necessita	**NA** – ***N**umber of **A**ssociations* (MA C.74)

Tabela C.78: Métrica Auxiliar TND

Nome	**TND** – ***T**otal **N**umber of **D**ependencies*
Definição informal	Número total de ligações de dependência no modelo SD/SR

Definição formal	`context ISTAR`
	`def:TND():Double = self.hasNode ->`
	`  select(n:Node | n.oclIsKindOf(Actor)) ->`
	`  iterate(n:Node; total:Integer = 0 |`
	`  let nd:Integer = n.oclAsType(Actor).ND() in total + nd)`

Necessita	**ND** – ***N**umber of **D**ependencies* (MA C.36)

Tabela C.79: Métrica Auxiliar TNC

Nome	**TNC** – ***T**otal **N**umber of **C**ontributions*
Definição informal	Número total de ligações de contribuição no modelo SD/SR

Definição formal	`context ISTAR`
	`def:TNC():Integer = self.hasNode ->`
	`  select(n:Node | n.oclIsKindOf(Actor)) ->`
	`  iterate(n:Node; total:Integer = 0 |`
	`  let ndsi:Integer = n.oclAsType(Actor).NDSI() in`
	`  total + ndsi)`

Necessita	**NDSI** – ***N**umber of **D**ecompositions of a **S**oftgoal **I**nside* (MA C.9)

Tabela C.80: Métrica Auxiliar TNDL

Nome	**TNDL** – ***T**otal **N**umber of **D**ecomposition **L**inks*
Definição informal	Número total de ligações de decomposição no modelo SD/SR

Definição formal	`context ISTAR`
	`def:TNDL():Integer = self.hasNode ->`
	`  select(n:Node | n.oclIsKindOf(Actor)) ->`
	`  iterate(n:Node; total:Integer = 0 |`
	`  let ndti:Integer = n.oclAsType(Actor).NDTI() in`
	`  total + ndti)`

Necessita	**NDTI** – ***N**umber of **D**ecompositions of a **T**ask **I**nside* (MA C.15)

Aplicação da Ferramenta aos Casos de Estudo

Neste anexo são apresentados os resultados da aplicação da ferramenta aos casos de estudo introduzidos na secção 6.1. Para cada caso de estudo, são apresentadas 2 (duas) figuras: uma que representa o modelo do sistema de *software*, e outra com o valor das métricas calculadas, assim como erros detetados pela ferramenta, avisos e sugestões.

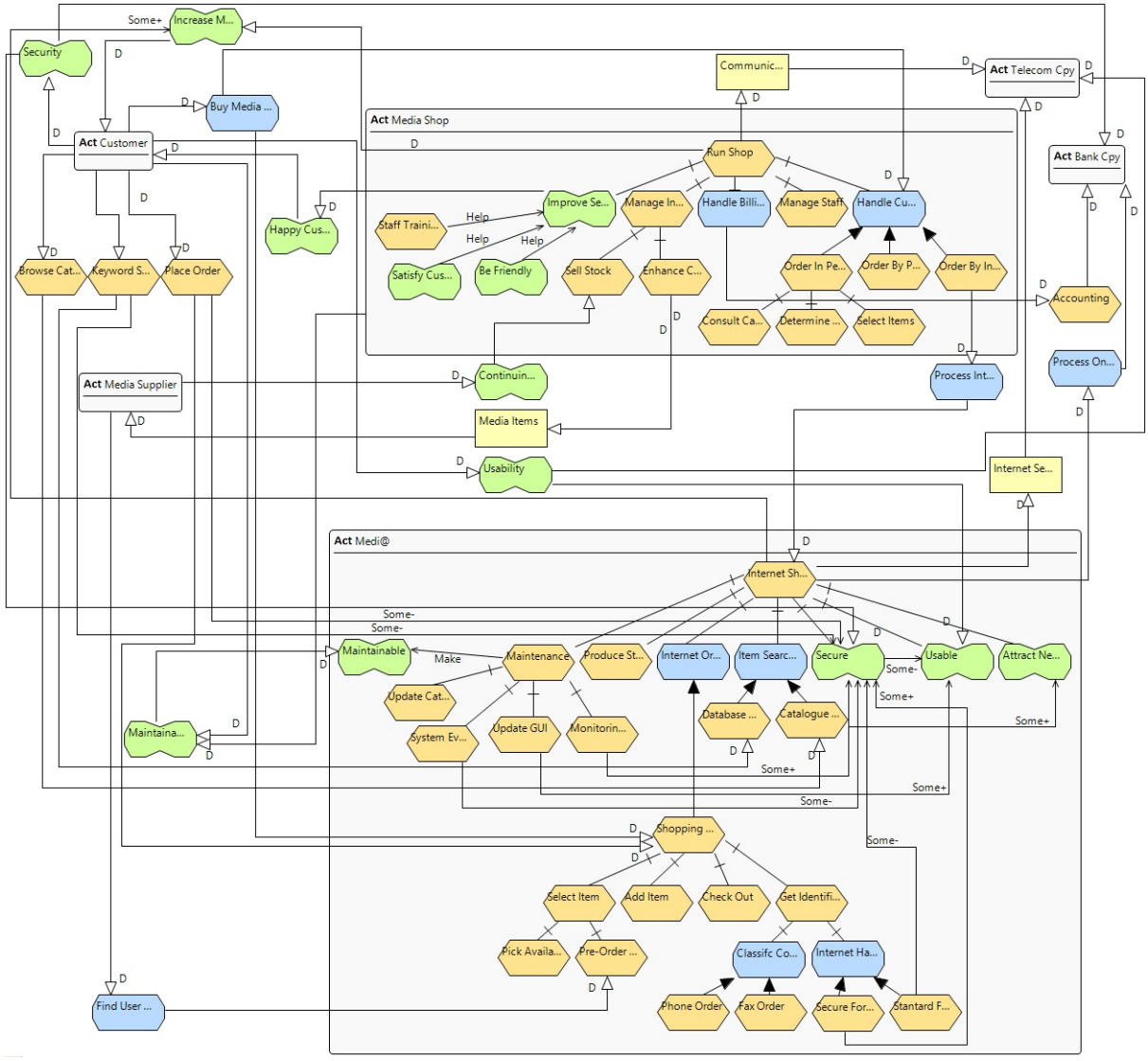


Figura D.1: Sistema *Media Shop* modelado pela ferramenta

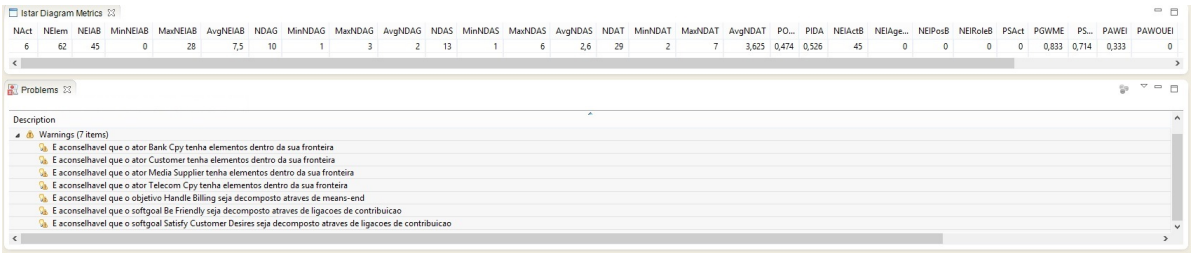


Figura D.2: Métricas e erros/avisos do modelo do sistema *Media Shop*

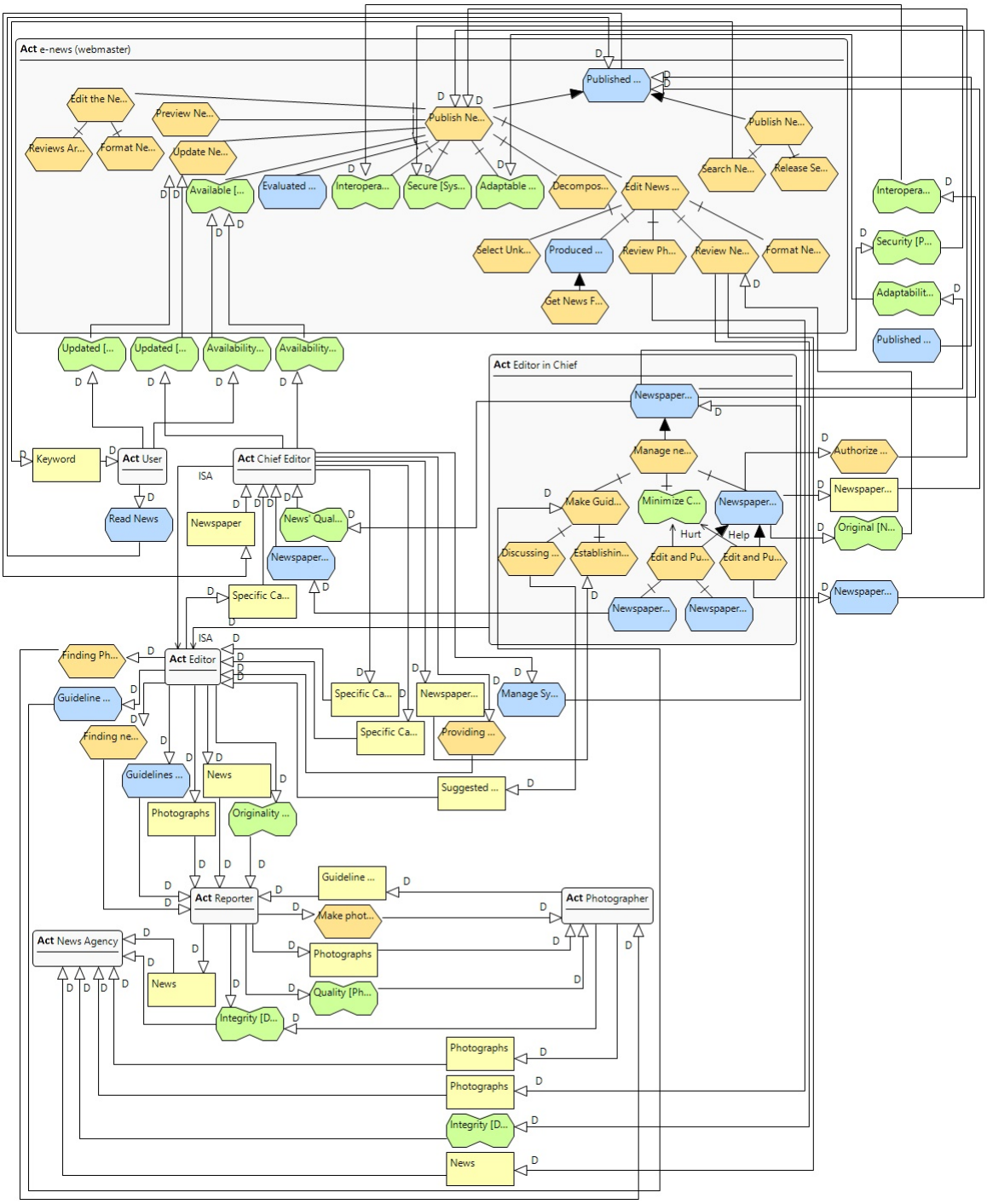


Figura D.3: Sistema *Newspaper Office* modelado pela ferramenta

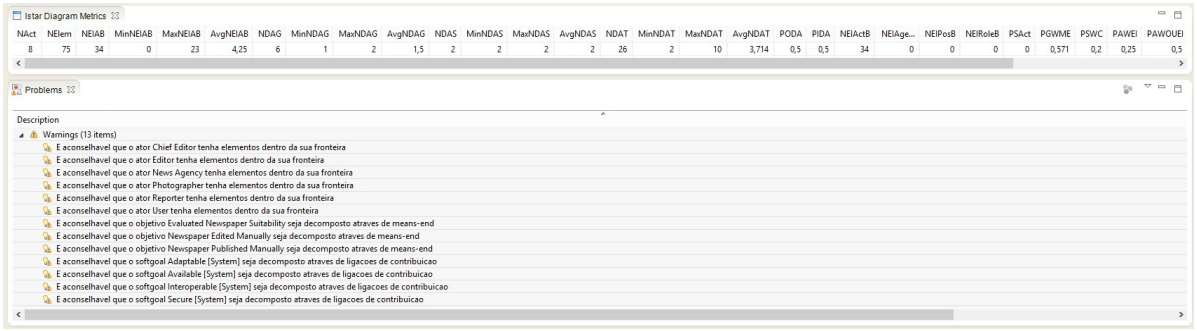


Figura D.4: Métricas e erros/avisos do modelo do sistema *Newspaper Office*

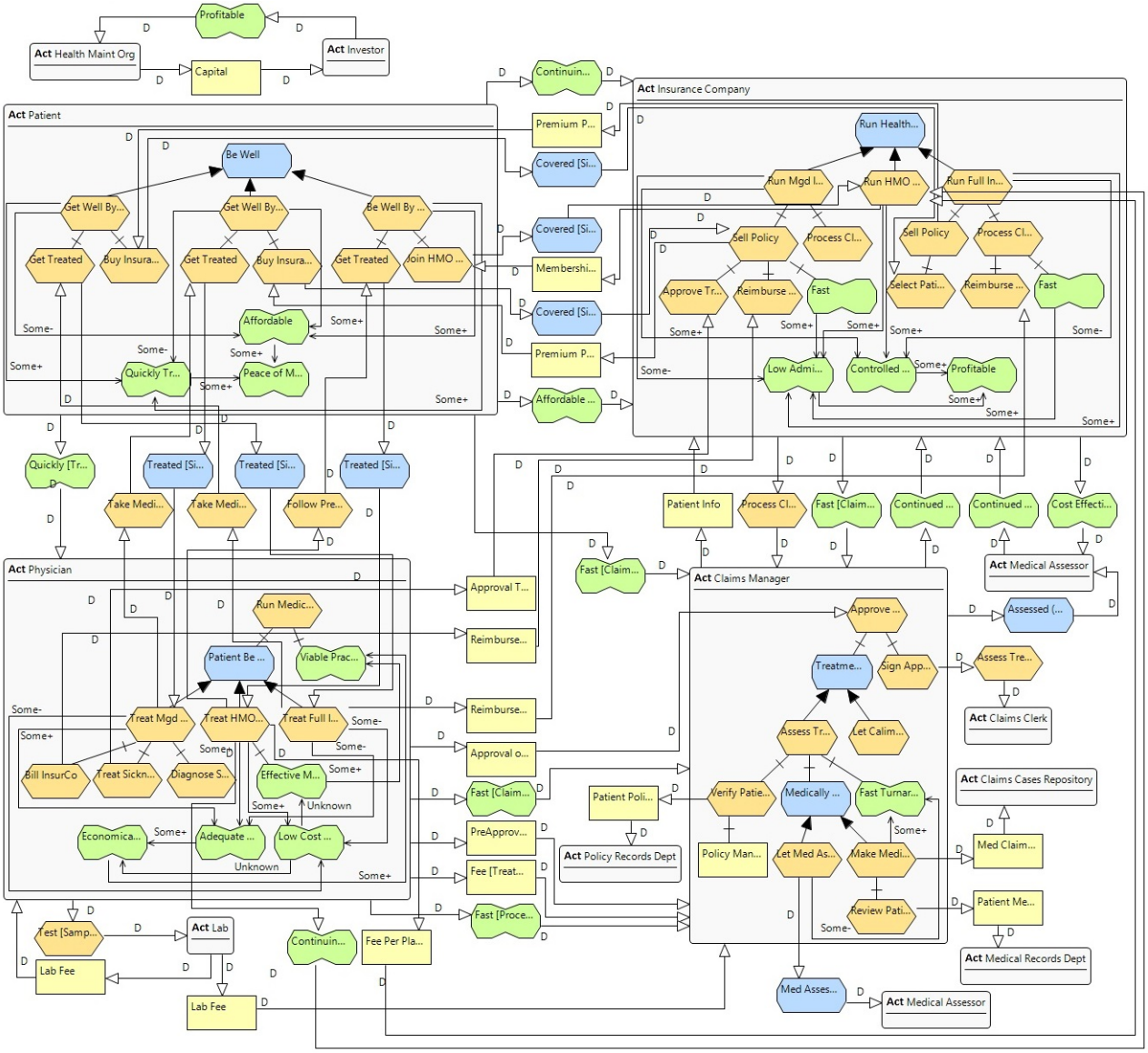


Figura D.5: Sistema *Health Care* modelado pela ferramenta

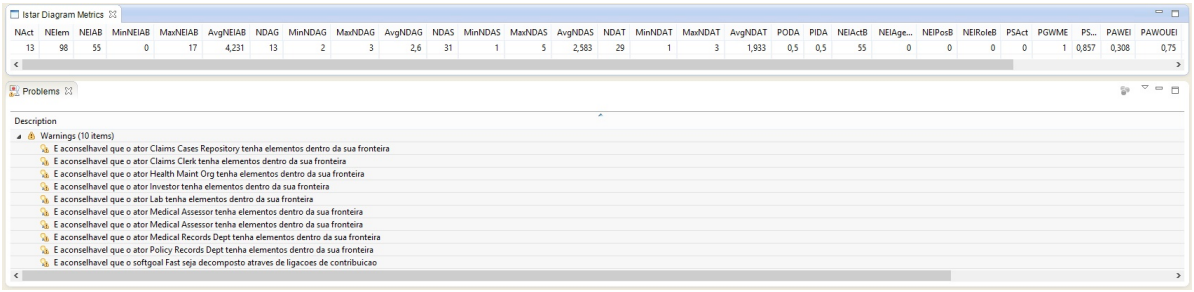


Figura D.6: Métricas e erros/avisos do modelo do sistema *Health Care*

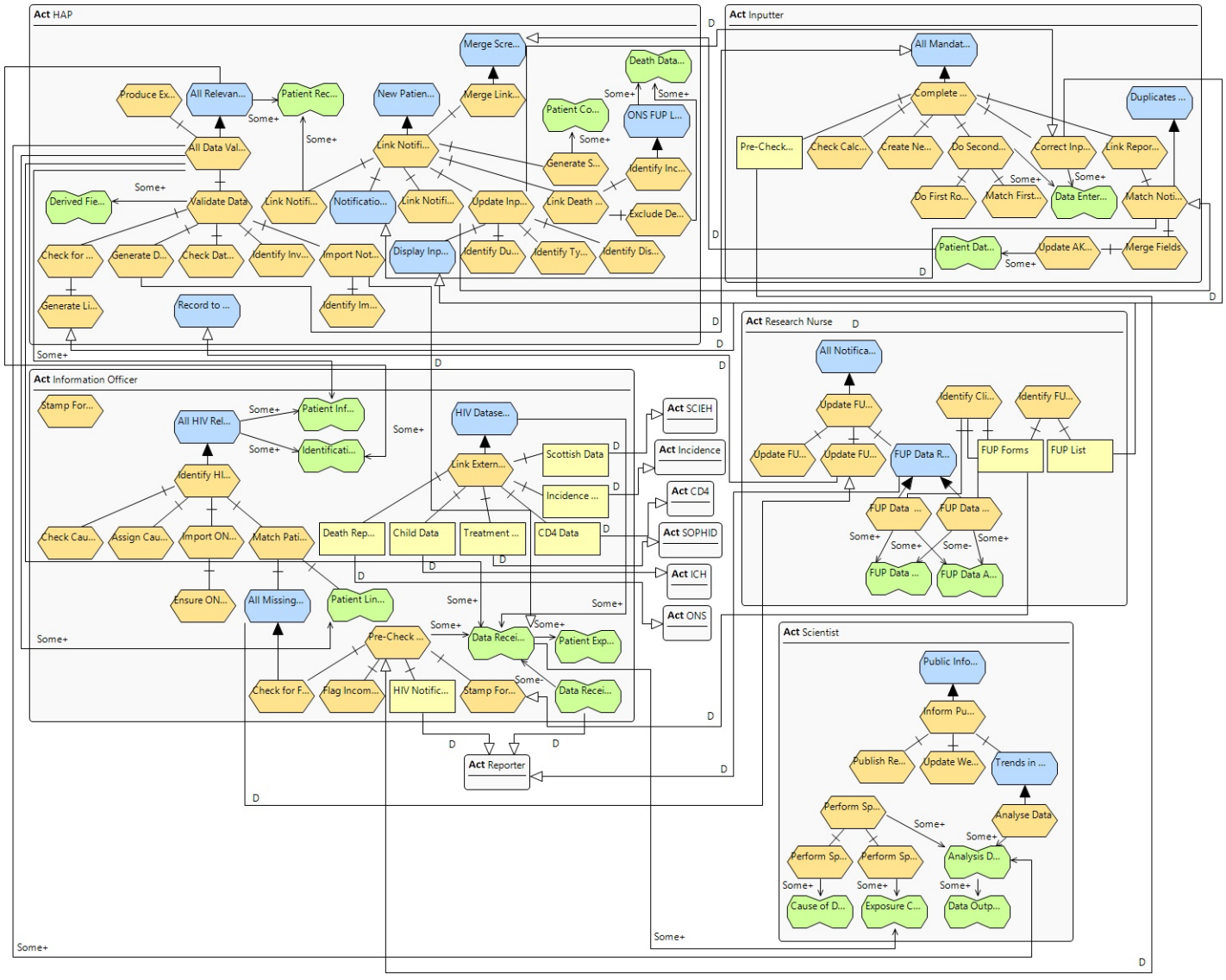


Figura D.7: Sistema *Health Protection Agency* modelado pela ferramenta

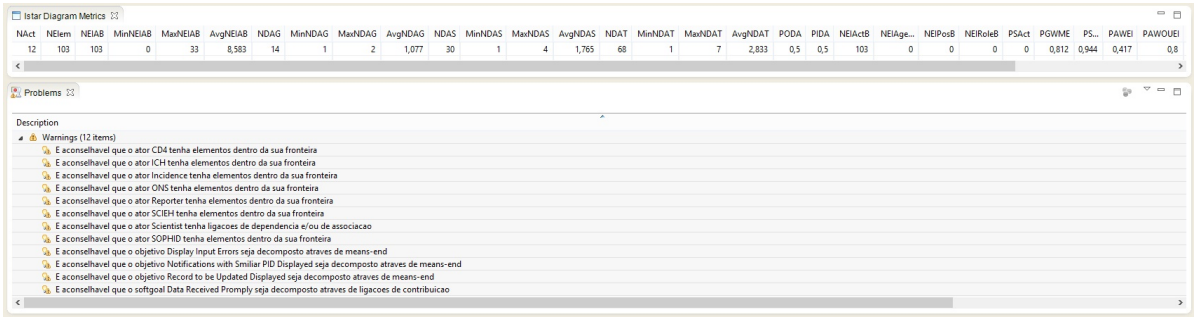


Figura D.8: Métricas e erros/avisos do modelo do sistema *Health Protection Agency*

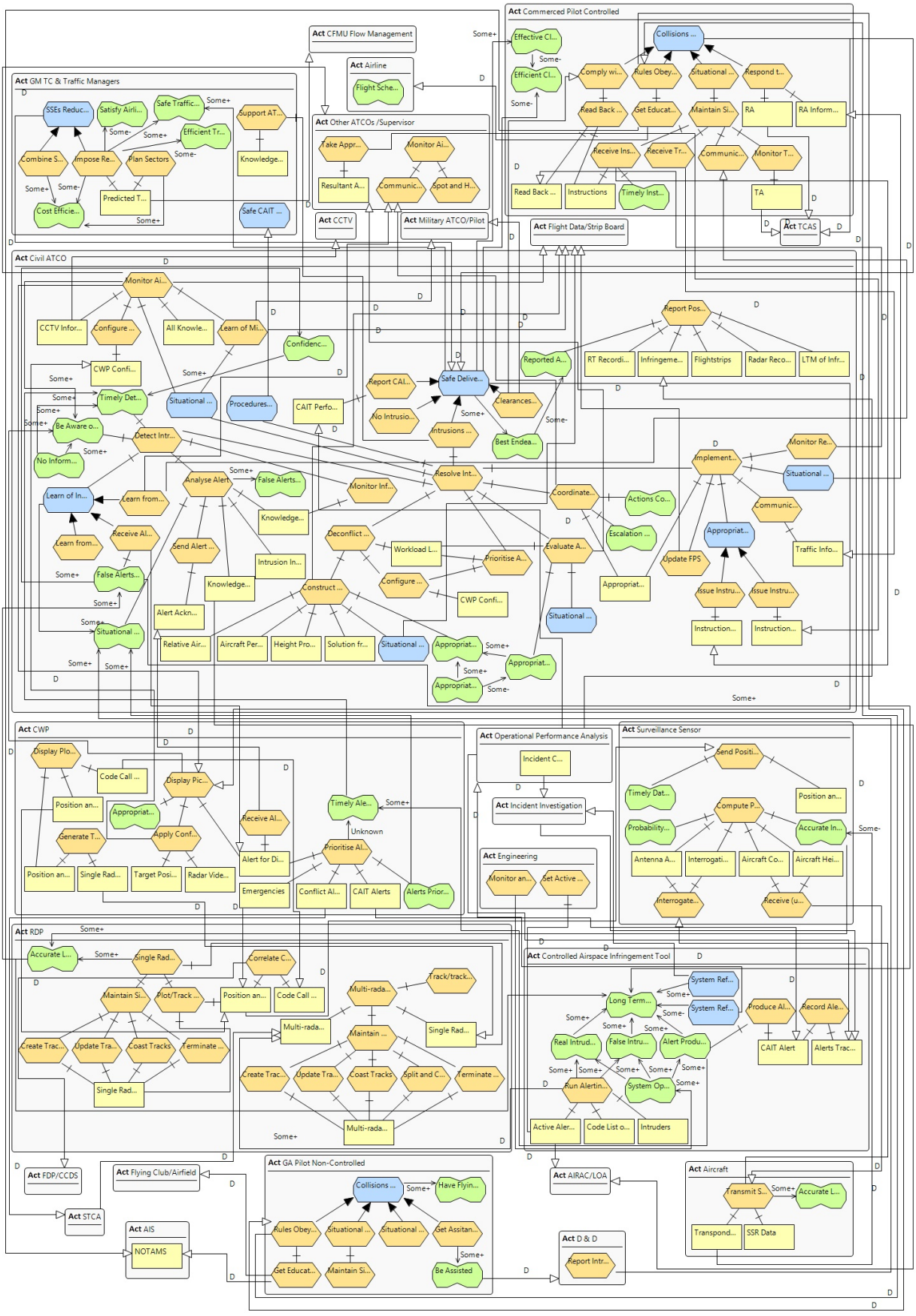


Figura D.9: Sistema *National Air Traffic Services* modelado pela ferramenta

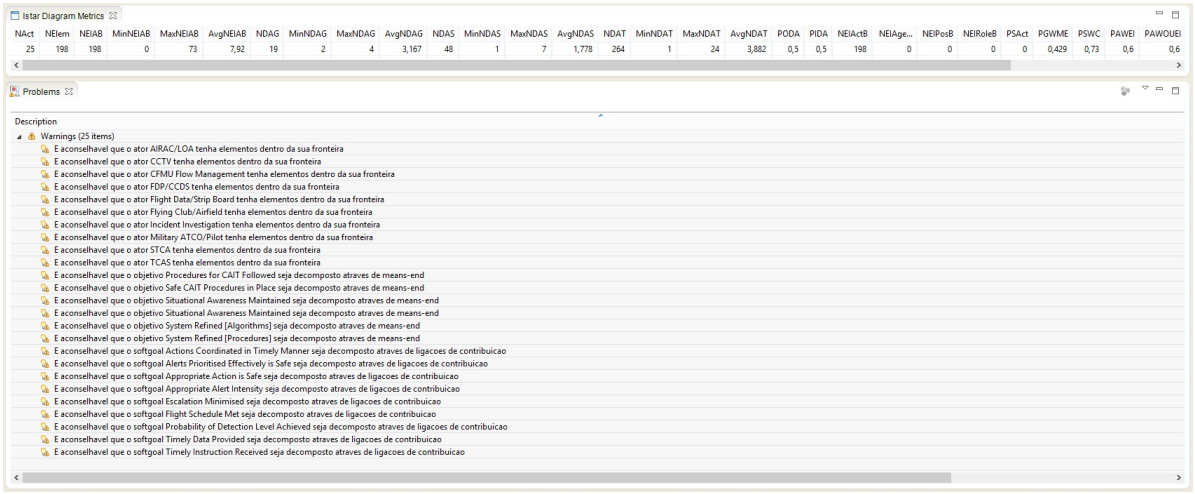


Figura D.10: Métricas e erros/avisos do modelo do sistema *National Air Traffic Services*

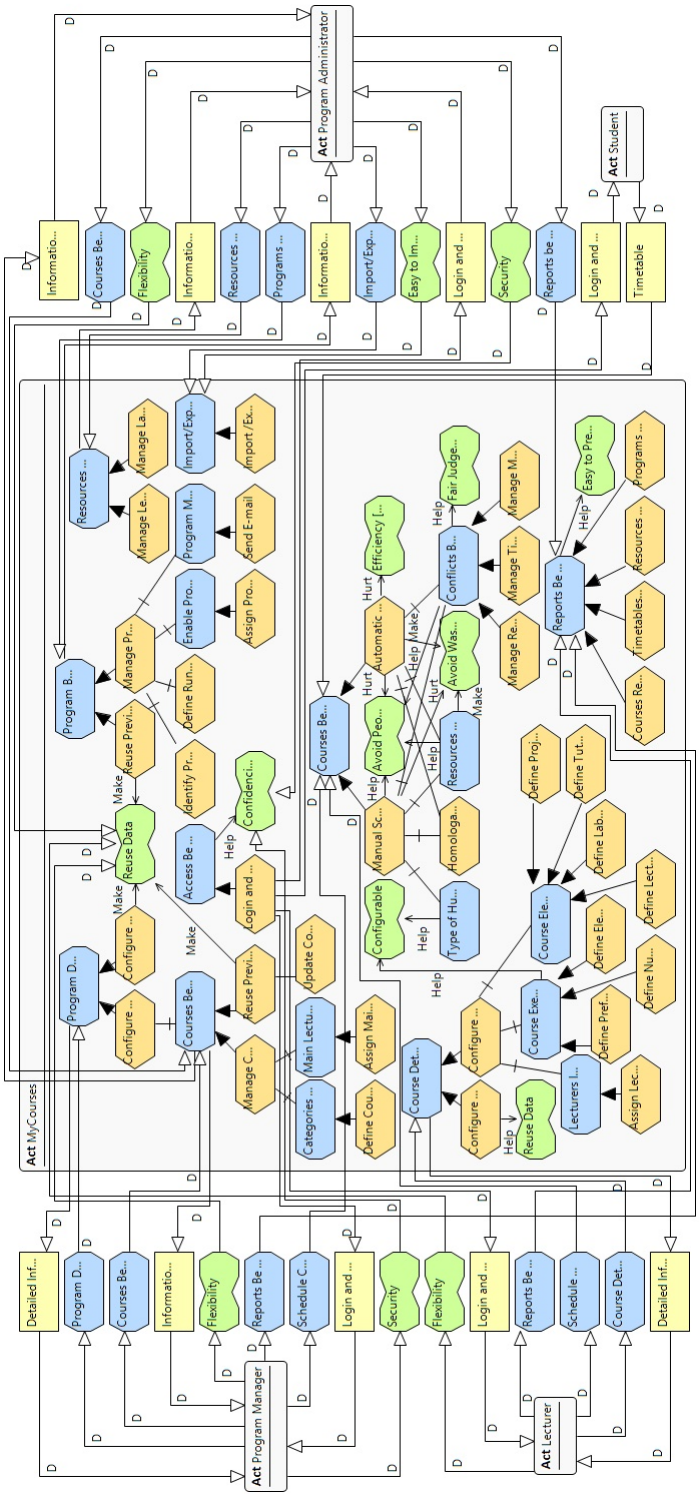


Figura D.11: Sistema *My Courses* modelado pela ferramenta

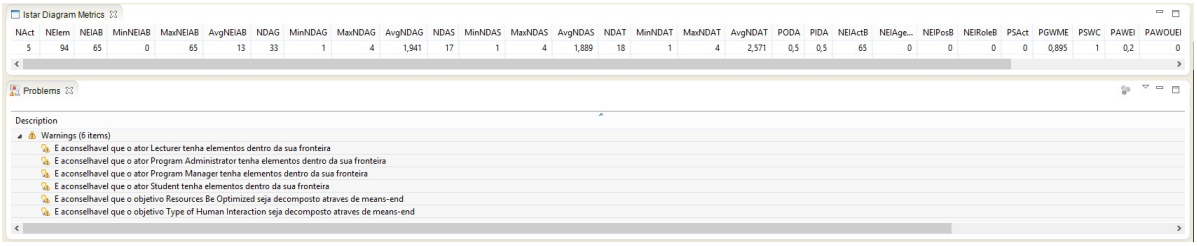


Figura D.12: Métricas e erros/avisos do modelo do sistema *My Courses*

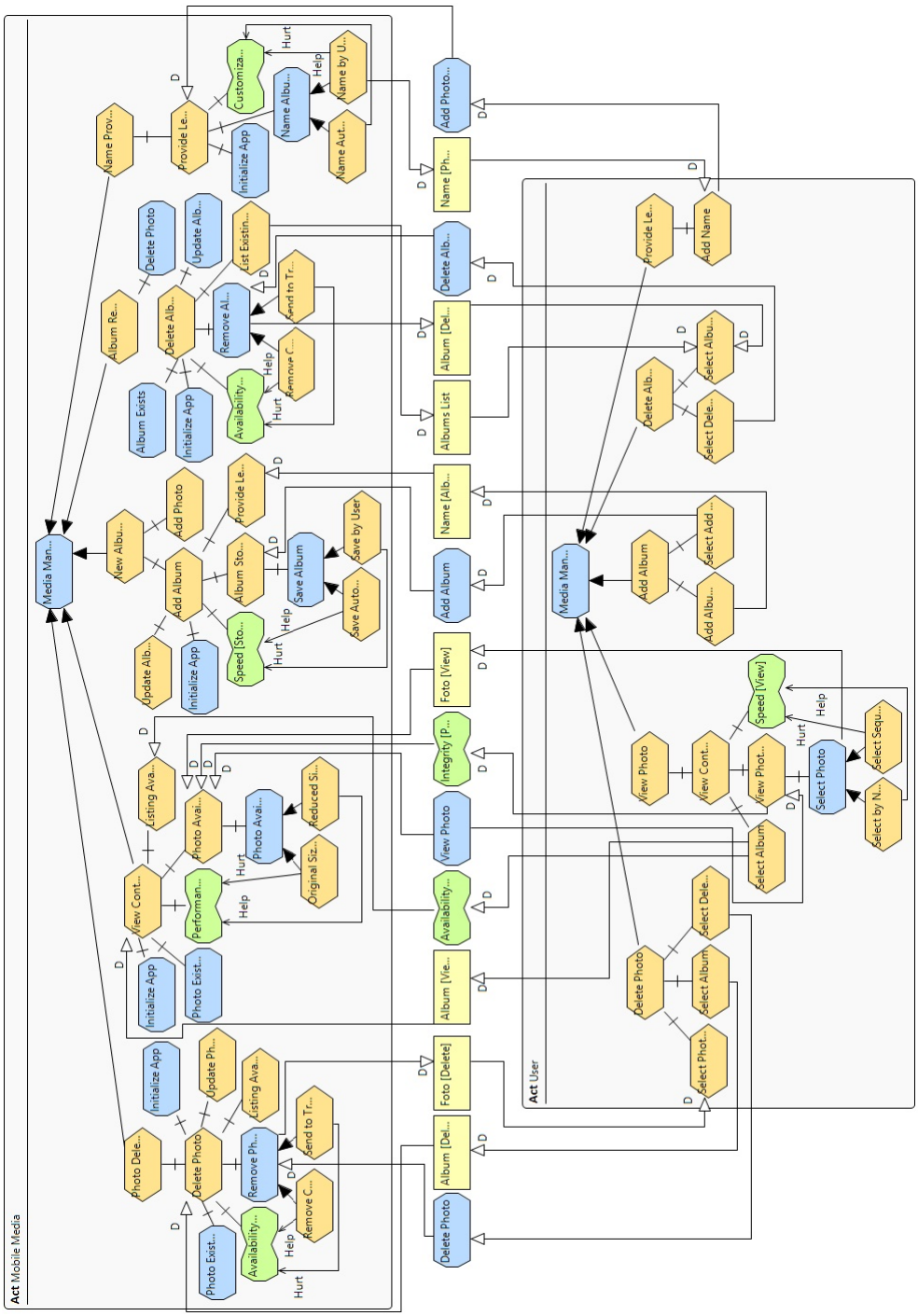


Figura D.13: Sistema *Mobile Media* modelado pela ferramenta

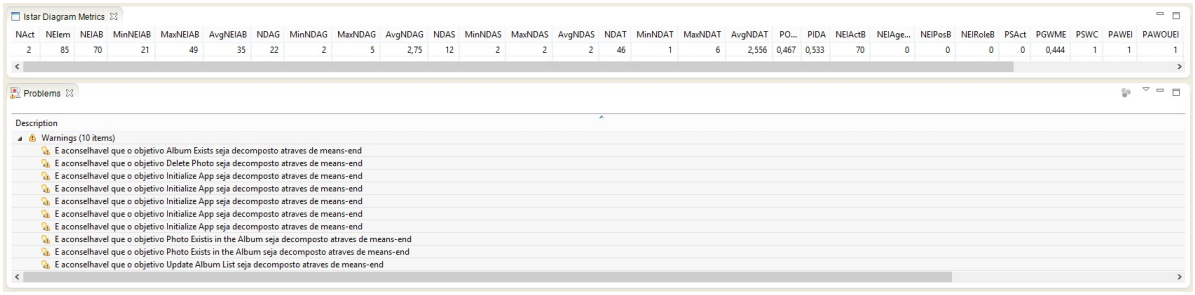


Figura D.14: Métricas e erros/avisos do modelo do sistema *Mobile Media*

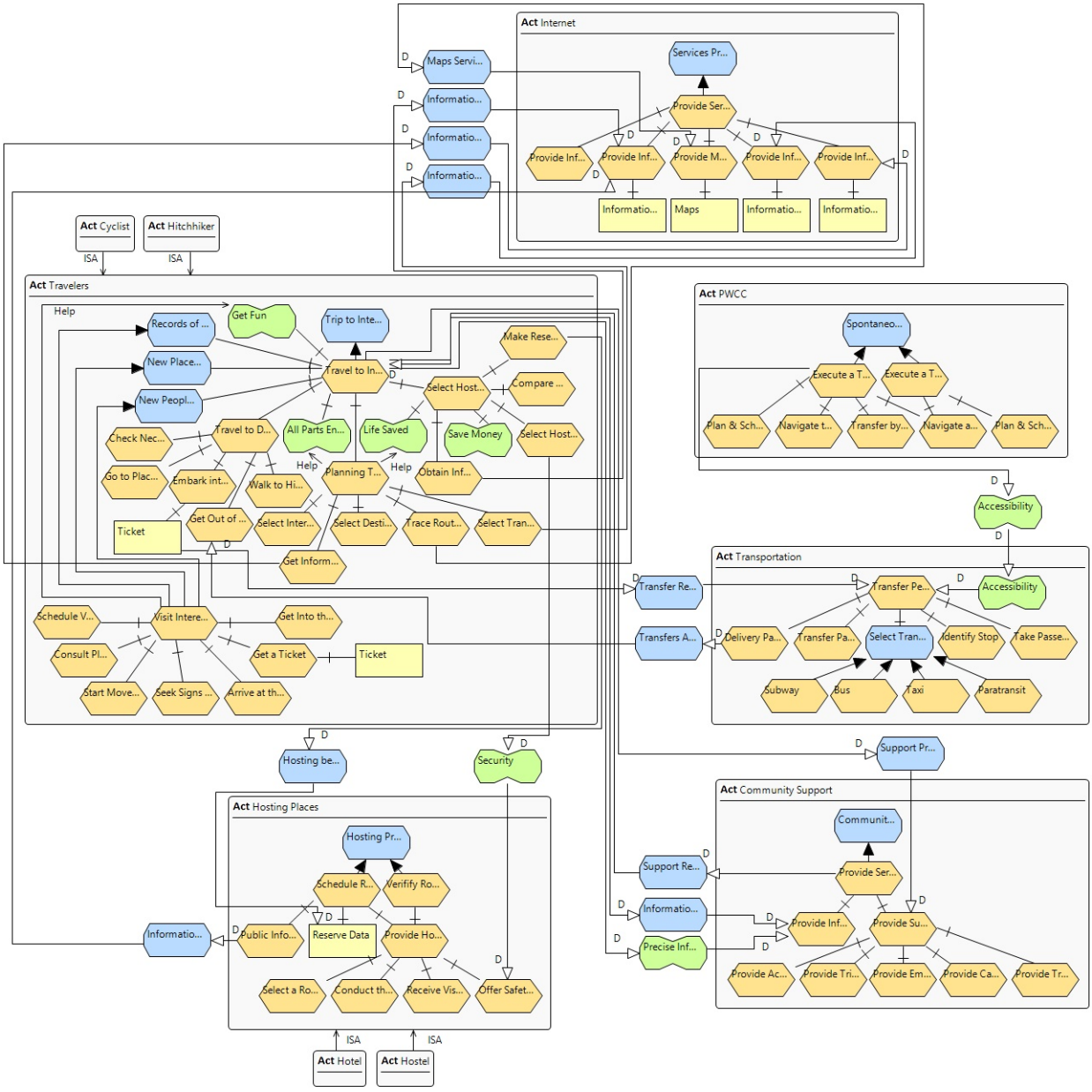


Figura D.15: Sistema *By The Way* modelado pela ferramenta

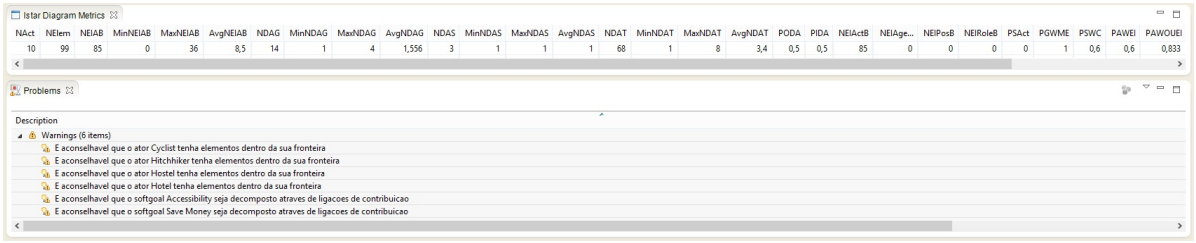


Figura D.16: Métricas e erros/avisos do modelo do sistema *By The Way*

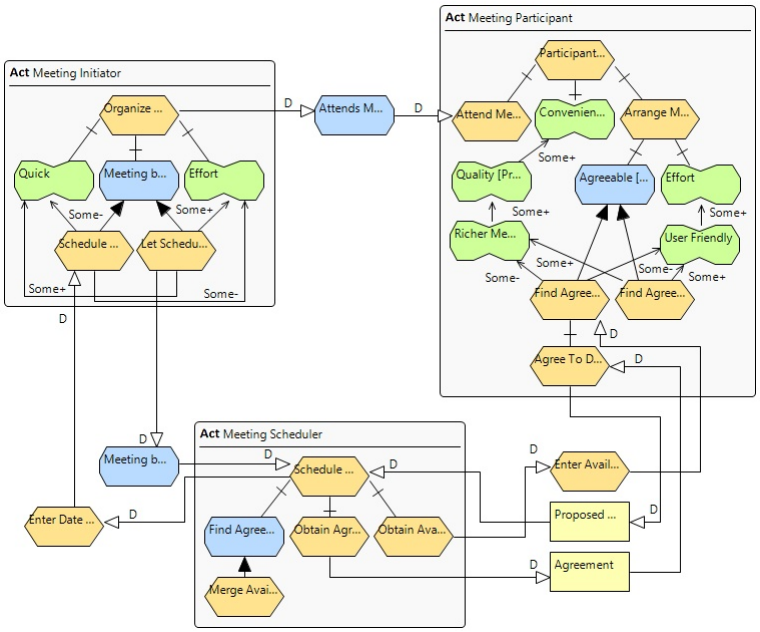


Figura D.17: Sistema *Meeting Scheduler* modelado pela ferramenta

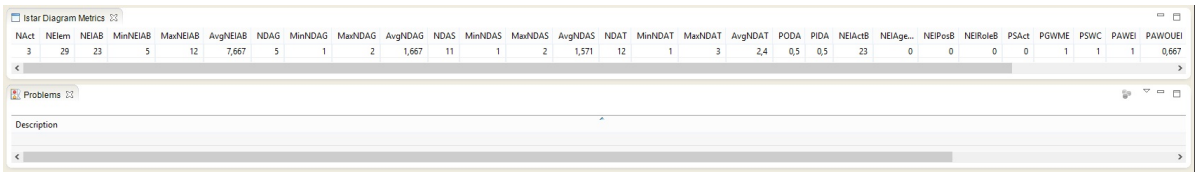


Figura D.18: Métricas e erros/avisos do modelo do sistema *Meeting Scheduler*

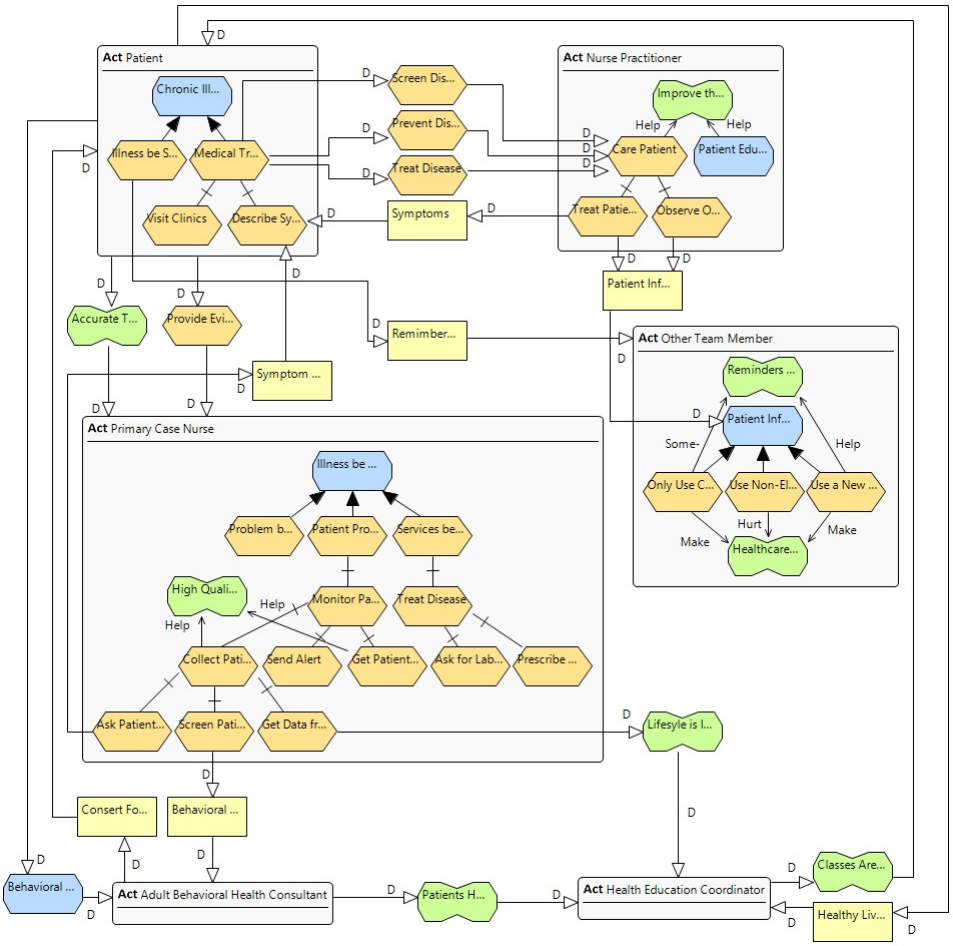


Figura D.19: Sistema *Patient Wellness Tracking* modelado pela ferramenta

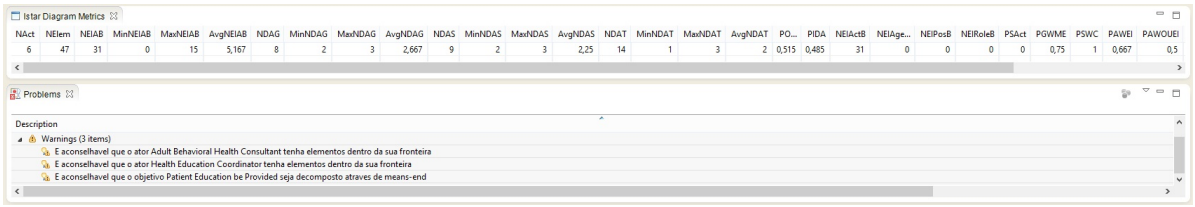


Figura D.20: Métricas e erros/avisos do modelo do sistema *Patient Wellness Tracking*

E

Adaptação dos Casos de Estudo

Neste anexo são apresentadas as adaptações efetuadas a cada um dos casos de estudo. Dentro de cada um dos sistemas, as alterações estão organizadas por ator, sendo apresentada uma tabela com as diferenças entre o caso de estudo original e a sua modelação com a ferramenta.

Algumas alterações são comuns a todos os casos de estudo, como é o caso da nomenclatura das ligações de contribuição. Assim, em todos os sistemas em que tal se verifique, assumiu-se que:

- Ligações ++ correspondem a ligações de contribuição *Make*
- Ligações + correspondem a ligações de contribuição *Some+*
- Ligações – correspondem a ligações de contribuição *Some-*
- Ligações – – correspondem a ligações de contribuição *Break*

E.1 Diferenças no Sistema *Media Shop*

Por utilizar o estilo *joint-venture* [Fux+01; KGM02], o sistema apresenta 6 (seis) atores presentes na fronteira de outros atores. Como os atores internos não foram detalhados ou, quando o foram, apresentaram novamente atores dentro da sua fronteira, optou-se por não se apresentar a sua representação no modelo.

E.1.1 Ator *Media Shop*

Tabela E.1: Erros e respetivas correções no ator *Media Shop*

Erro no modelo original	Correção efetuada
Softgoal ***Improve Service*** decomposto através de ligações de decomposição	Ligações de contribuição *Help*

E.1.2 Ator *Medi@*

Tabela E.2: Erros e respetivas correções no ator *Medi@*

Erro no modelo original	Correção efetuada
Tarefa ***Select Item*** decomposta através de ligações *means-end*	Ligações de decomposição
Tarefa ***Get Identification Detail*** decomposta através de ligações *means-end*	Ligações de decomposição

E.2 Diferenças no Sistema *Newspaper Office*

Por utilizar o estilo *joint-venture*, o sistema apresenta um ator, *System*, que contém todos os outros atores dentro da sua fronteira. Esse ator foi ignorado aquando da modelação do sistema. Adicionalmente, o ator ***e-news*** é também apresentado como ***webmaster***, com os mesmo elementos internos e as mesmas ligações, pelo que se assumiu que se tratava do mesmo ator, tendo sido representado com o nome ***e-news (webmaster)***.

E.3 Diferenças no Sistema *Health Care*

Com exceção de alterações referentes à nomenclatura das ligações de contribuição, não foram efetuadas alterações ao caso de estudo original.

E.4 Diferenças no Sistema *Health Protection Agency*

E.4.1 Ator *Research Nurse*

Tabela E.3: Erros e respetivas correções no ator *Research Nurse*

Erro no modelo original	Correção efetuada
Tarefa ***Identify Clinician Details*** decomposta através de ligações *means-end*	Ligações de decomposição
Tarefa ***Identify FUP Forms*** decomposta através de ligações *means-end*	Ligações de decomposição

E.5 Diferenças no Sistema *National Air Traffic Services*

E.5.1 Ator *Aircraft*

Tabela E.4: Erros e respetivas correções no ator *Aircraft*

Erro no modelo original	Correção efetuada
Tarefa ***Transmit SSR Data*** decomposta através da ligação *means-end*	Ligação de decomposição

E.5.2 Ator *GA Pilot Non-Controlled*

Tabela E.5: Erros e respetivas correções no ator *GA Pilot Non-Controlled*

Erro no modelo original	Correção efetuada
Softgoal ***Be Assisted*** decomposto através de ligação *means-end*	Ligação de contribuição *Some+*

E.6 Diferenças no Sistema *My Courses*

Com exceção de alterações referentes à nomenclatura das ligações de contribuição, não foram efetuadas alterações ao caso de estudo original.

E.7 Diferenças no Sistema *Mobile Media*

Com exceção de alterações referentes à nomenclatura das ligações de contribuição, não foram efetuadas alterações ao caso de estudo original.

E.8 Diferenças no Sistema *By The Way*

Com exceção de alterações referentes à nomenclatura das ligações de contribuição, não foram efetuadas alterações ao caso de estudo original.

E.9 Diferenças no Sistema *Meeting Scheduler*

Com exceção de alterações referentes à nomenclatura das ligações de contribuição, não foram efetuadas alterações ao caso de estudo original.

E.10 Diferenças no Sistema *Patient Wellness Tracking*

Com exceção de alterações referentes à nomenclatura das ligações de contribuição, não foram efetuadas alterações ao caso de estudo original.